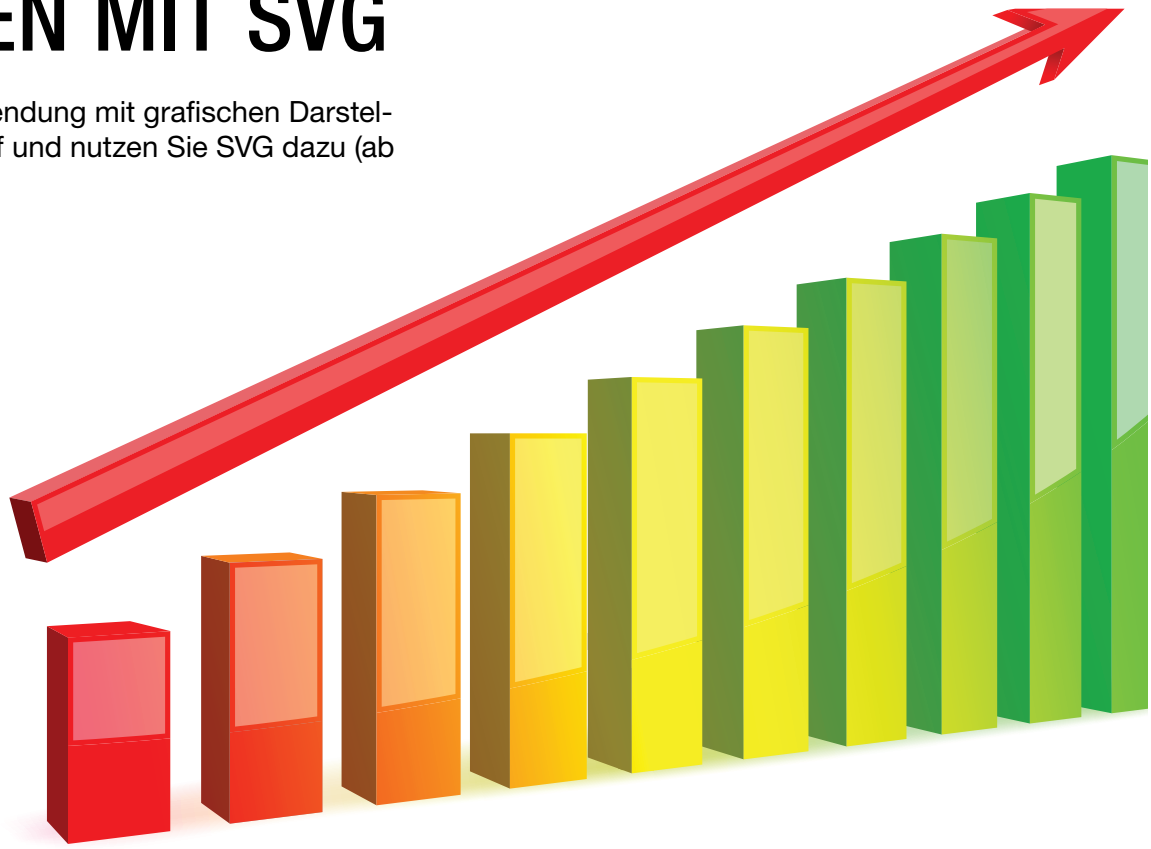
HAUFE.
Ausgabe 03/2018

ACCESS IM UNTERNEHMEN

GRAFIKEN MIT SVG

Peppen Sie Ihre Anwendung mit grafischen Darstellungen Ihrer Daten auf und nutzen Sie SVG dazu (ab S. 37)



In diesem Heft:

VEREINSVERWALTUNG: FORMULARE, TEIL 1

Entwickeln Sie die Benutzeroberfläche für die Vereinsverwaltung.

SEITE 66

REFACTORING

Passen Sie die übrigen Elemente einer Anwendung an neue Objekt- und Feldnamen an.

SEITE 55

EINFACHE REIHENFOLGE

Lernen Sie eine einfache Methode zum Festlegen einer individuellen Reihenfolge von Daten kennen.

SEITE 13

Ein Bild sagt tausend Worte

Die grafische Darstellung von Sachverhalten unter Access wird etwas stiefmütterlich behandelt. Es gibt keine wirklich brauchbare und einfach bedienbare Möglichkeit, etwa schnell ein benutzerdefiniertes Diagramm auf das Formular oder in den Bericht zu zaubern. Dies wollen wir ändern und liefern in dieser Ausgabe den ersten Beitrag über die Erstellung von Grafiken über SVG (Scalable Vector Graphics).

In diesem Beitrag namens **SVG als Grafikkomponente**, den Sie ab Seite 37 finden, liefert Sascha Trowitzsch zuerst die Grundlagen zum Erstellen grafischer Objekte und zeigt dann, wie Sie ein Diagramm erstellen, dessen Elemente sogar anklickbar sind. Dort zeigen wir beispielsweise den Gesamtumsatz nach Ländern an und erlauben per Mausklick den Zugriff auf die ID des jeweiligen Landes. Das können Sie leicht ausbauen, indem Sie etwa ein Detailformular mit weiteren Informationen zu den in diesem Land getätigten Umsätzen anzeigen. Seien Sie auch gespannt auf die Fortsetzung in der nächsten Ausgabe!

In der aktuellen Ausgabe führen wir außerdem unsere Reihe zum Thema Vereinsverwaltung fort. Nachdem wir in Ausgabe 6/2017 gezeigt haben, wie Sie das Datenmodell für die Vereinsverwaltung herleiten und die Daten von Excel nach Access migrieren, gehen wir in dieser Ausgabe einen Schritt weiter. Unter dem Titel **Vereinsverwaltung: Formulare, Teil 1** zeigen wir ab Seite 66, wie Sie das Detailformular zur Anzeige von Mitgliedern aufbauen.

Der Beitrag **Zeiträume per Listenfeld und InputBox** zeigt ab Seite 28, wie Sie Zeiträume wie etwa Vereinszugehörigkeiten in einem Listenfeld verwalten können. Die Details zu diesen Zeiträumen zeigen wir zum Bearbeiten diesmal nicht in einem eigenen Formular an, sondern liefern mit der **InputBox**-Funktion eine schlankere Möglichkeit. Außerdem zeigt dieser Beitrag, wie Sie prüfen können, ob neue Zeiträume zu den vorhandenen Zeiträumen passen und sich nicht etwa damit überschneiden.

Darüber hinaus beschäftigen wir uns mit dem Löschen von Datensätzen in Formularen – genauer gesagt mit den dabei ausgelöste Ereignissen. Es gibt nämlich nicht nur ein Ereignis, das beim Löschen ausgelöst wird, sondern in der Regel gleich drei. Mit diesen können Sie den Löschvorgang beeinflussen und beispielsweise die eingebauten Meldungen zur Löschbestätigung durch benutzerdefinierte Meldungen ersetzen. Wie das gelingt, erfahren Sie unter dem Titel **Löschen in Formularen: Ereignisse** ab Seite 2. Ein weiterer Beitrag rund um das Löschen von Daten, diesmal mit dem Listenfeld als Mittelpunkt, startet unter dem Titel **Löschen im Listenfeld** auf Seite 22. Hier erfahren Sie, wie Sie ein Listenfeld um Funktionen erweitern, mit denen der Benutzer einen oder mehrere Einträge einfach mit der **Entf**-Taste löschen kann.

Der Beitrag **Reihenfolge einfach festlegen** liefert ab Seite 13 eine kleine Lösung, mit der Sie einem Formular in der Datenblatt- oder Endlosansicht eine einfache Funktion zum Sortieren der angezeigten Daten hinzufügen können.

Und schließlich kümmern wir uns noch um das Refactoring nach dem Umbenennen von Objekt- und Feldnamen: Welche Schritte erledigt Access automatisch, wenn Sie etwa einen Tabellennamen geändert haben und welche müssen Sie selbst noch durchführen, damit der Rest der Anwendung an die neue Tabellenbezeichnung angepasst wird? Danke an dieser Stelle an Lorenz Hölscher für die Anregung!

Und nun: Viel Spaß beim Lesen!

Ihr André Minhorst

Löschen in Formularen: Ereignisse

Das Löschen von Datensätzen in einem Formular ist eigentlich kein großes Problem: Man markiert den Datensatz über den Datensatzmarkierer und klickt auf die Entfernen-Taste. Manch ein Benutzer verzweifelt vielleicht daran, den Datensatz über den Datensatzmarkierer zu selektieren, weshalb er ihn dann nicht löschen kann – dann baut man ihm halt eine Schaltfläche, die auch den Datensatz löscht, der aktuell den Fokus hat. Was aber, wenn wir noch Aktionen durchführen wollen, bevor der Datensatz gelöscht wird – beispielsweise das Archivieren des Datensatzes oder das Ausführen weiterer Aktionen nach dem Löschen? Wie das funktioniert und was Sie beachten müssen, zeigt der vorliegende Beitrag.

Als Beispiel greifen wir die Datenbank aus dem Beitrag **Reihenfolge einfach festlegen** auf (**www.access-im-unternehmen.de/1129**). Hier verwenden wir ein Formular mit einem Unterformular, um die Daten der Tabelle **tblArtikel** in der Datenblattansicht anzuzeigen.

Ereignisse beim Löschen

Beim Löschen eines Datensatzes in einem Formular werden verschiedene Ereignisse ausgelöst. Wenn wir im Eigenschaftsfenster des Unterformulars zum Bereich Ereignis wechseln, finden wir einige Ereignisse, die möglicherweise in Zusammenhang mit dem Löschen eines Datensatzes ausgelöst werden. Genau das wollen wir untersuchen, also legen wir für die in Bild 1 mit dem Wert **[Ereignisprozedur]** versehenen Eigenschaften entsprechende Ereignisprozeduren an.

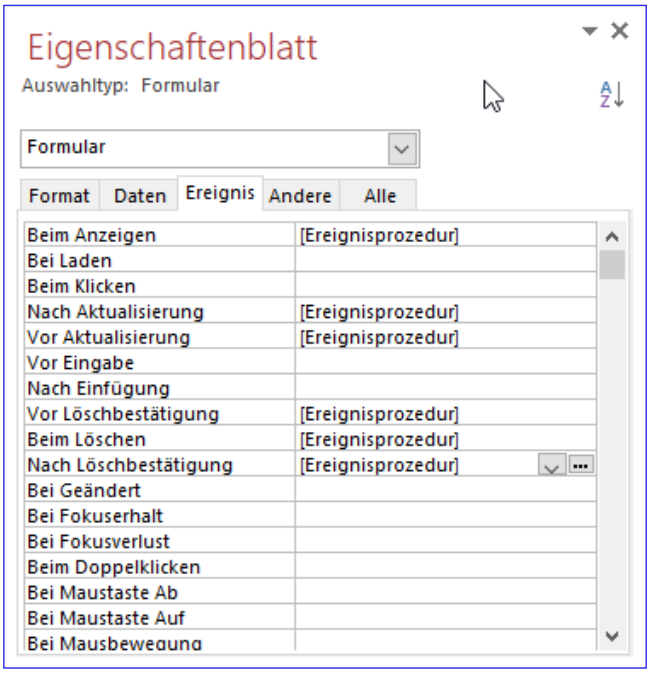


Bild 1: Ereignisse, die möglicherweise beim Löschen ausgelöst werden

Die dadurch automatisch im Klassenmodul des entsprechenden Formulars angelegten Ereignisprozeduren statten wir mit jeweils einer **Debug.Print**-Anweisung aus, welche lediglich den Namen der Ereignisprozedur im Direktfenster ausgibt.

Die Prozeduren sehen dann wie folgt aus:

```
Private Sub Form_AfterDelConfirm(Status As Integer)
    Debug.Print "AfterDelConfirm"
End Sub

Private Sub Form_AfterUpdate()
    Debug.Print "AfterUpdate"
End Sub

Private Sub Form_BeforeDelConfirm(Cancel As Integer, _
        Response As Integer)
    Debug.Print "BeforeDelConfirm"
End Sub
```

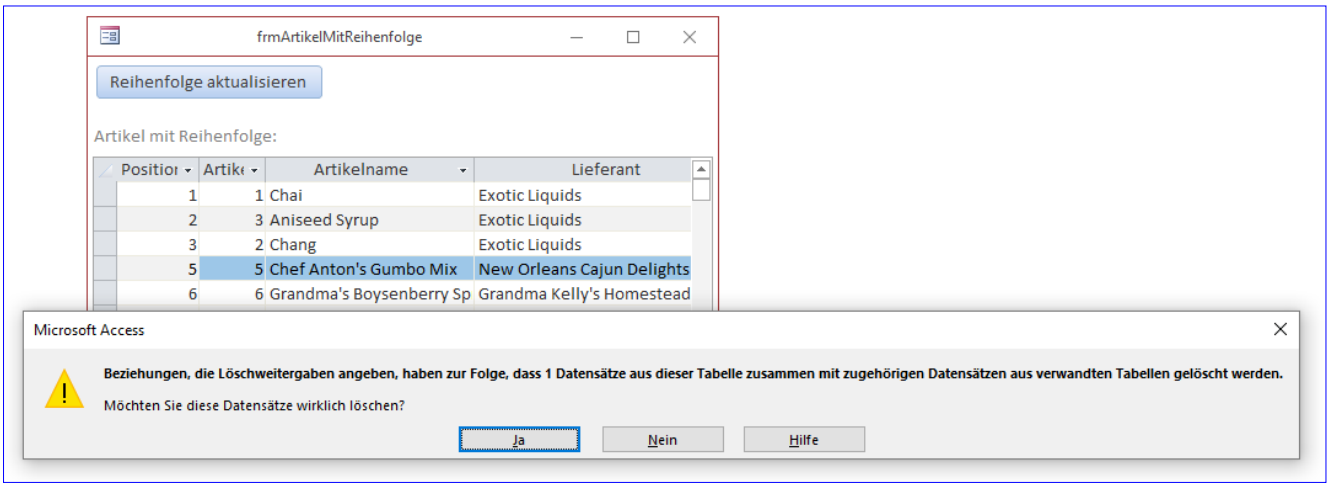


Bild 2: Meldung beim Löschen eines Datensatzes

```
Private Sub Form_BeforeUpdate(Cancel As Integer)
    Debug.Print "BeforeUpdate"
End Sub

Private Sub Form_Current()
    Debug.Print "Current"
End Sub

Private Sub Form_Delete(Cancel As Integer)
    Debug.Print "Delete"
End Sub
```

Wenn Sie nun einen Datensatz im Unterformular markieren und diesen mit der **Entf**-Taste oder dem entsprechenden Ribbon-Eintrag löschen, erscheint die Meldung aus Bild 2.

Diese Meldung besagt, dass das Löschen dieses Datensatzes das Löschen verknüpfter Datensätze (in diesem Fall aus der Tabelle **tblBestelldetails**) nach sich ziehen würde.

Wenn Sie einen neuen Datensatz in diesem Formular anlegen, der noch keine verknüpften Datensätze in der Tabelle **tblBestelldetails** aufweist und diesen löschen, erscheint eine andere Meldung – nämlich die aus Bild 3.

Zu diesem Zeitpunkt werfen wir auch einen Blick in den Direktbereich des VBA-Editors. Dieser sieht nun wie in Bild 4 aus. Wir merken uns den aktuellen Zustand und bestätigen die Löschmeldung mit einem Klick auf die Schaltfläche **Ja**.

Im Direktbereich sieht es dann wie folgt aus, wobei wir den Zeitpunkt der Anzeige der Löschmeldung hier durch einen entsprechenden Kommentar eingefügt haben:

```
Delete
Current
BeforeDelConfirm
'Meldung
AfterDelConfirm
```

Beim Löschen werden also vier Ereignisse ausgelöst:

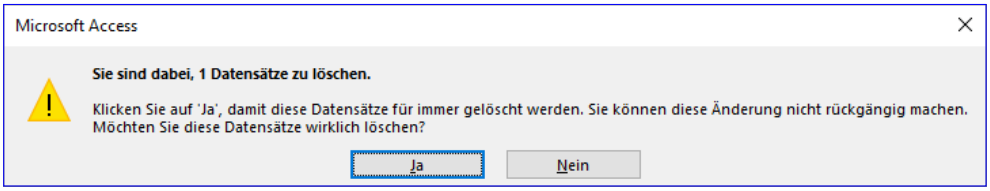


Bild 3: Meldung beim Löschen eines nicht mit anderen Datensätzen in Beziehung stehenden Datensatzes

- **Beim Löschen**
- **Beim Anzeigen**
- **Vor Löschbestätigung**
- **Vor Löschbestätigung**

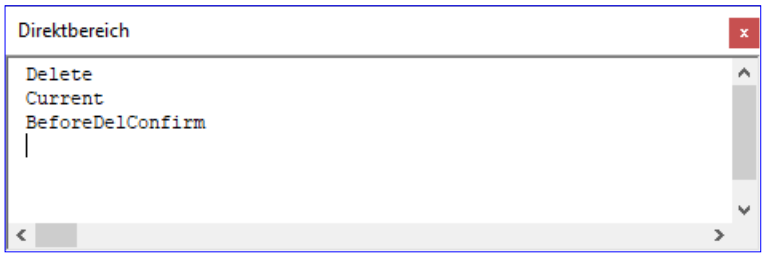


Bild 4: Der Direktbereich mitten in einem Löschvorgang

Wenn wir beim Erscheinen des Meldungsfensters nicht auf **Ja**, sondern auf **Nein** klicken, löst dies noch ein Ereignis mehr aus:

```
Delete
Current
BeforeDelConfirm
Current
AfterDelConfirm
```

Das **Current**-Ereignis (**Beim Anzeigen**) wird hier ausgelöst, weil der Datensatzzeiger wieder auf den zu löschenden Datensatz eingestellt wird.

Löschbestätigung deaktivieren

Aber gab es da nicht die Möglichkeit, die Löschbestätigung zu deaktivieren und Datensätze direkt zu löschen, wenn man auf die **Entf**-Taste klickt? Ja, die gibt es.

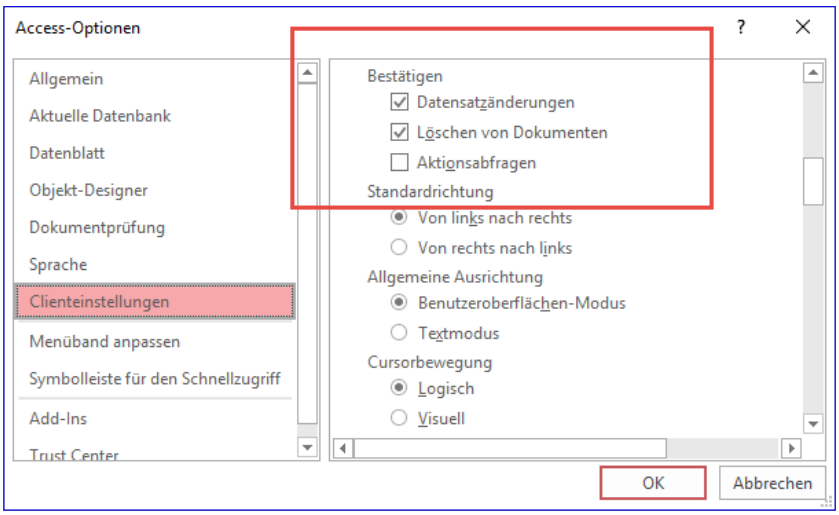


Bild 5: Meldung beim Löschen eines Datensatzes

Dazu öffnen Sie den Optionen-Dialog von Access und wechseln dort zum Bereich **Clienteinstellungen**. Hier finden Sie unter **Bestätigen** drei Optionen: **Datensatzänderungen**, **Löschen von Dokumenten** und **Aktionsabfragen** (siehe Bild 5). In unserem Fall benötigen wir die Option **Datensatzänderungen**, die wir deaktivieren.

Was geschieht nun im Direktbereich, wenn wir einen Datensatz löschen? Es werden nur noch die folgenden beiden Ereignisse ausgelöst:

```
Delete
Current
```

Wann immer wir die Ereignisse mit Prozeduren versehen möchten, müssen wir also bei **Vor Löschbestätigung** und **Nach Löschbestätigung** berücksichtigen, dass diese bei deaktivierter Option **Datensatzänderungen auf dem Client-Rechner** nicht ausgeführt werden.

Warnungen per VBA ausschalten

Wenn Sie die Warnmeldungen unabhängig von den Einstellungen auf dem Clientrechner abschalten wollen, etwa um keine Meldung anzuzeigen, können Sie die Meldungen auch per VBA deaktivieren.

Dazu rufen Sie einfach die Methode **SetWarnings** des **DoCmd**-Objekts mit dem Parameter **False** auf:

```
DoCmd.SetWarnings False
```

Reihenfolge einfach festlegen

Wenn Sie Daten mit individueller Reihenfolge anlegen wollen, können Sie das mit sehr wenig Aufwand anstellen, indem Sie dafür einfach ein Zahlenfeld anlegen. Für dieses kann der Benutzer dann einen eigenen Wert festlegen, der zwischen dem zweier vorhandener Felder liegt und die Reihenfolge-Werte dann per Mausklick aktualisieren. Wie das genau aussieht und wie einfach die Programmierung hierfür ist, lesen Sie im vorliegenden Beitrag.

Warum die Reihenfolge festlegen?

Gründe, um die Reihenfolge von Datensätzen in einer Tabelle festzulegen, gibt es genügend. In unserem Beispiel der Tabelle **tblArtikel** gehen wir zum Beispiel davon aus, dass der Benutzer die Artikel in einer bestimmten Reihenfolge für einen Katalog ausgeben möchte.

Feld zum Speichern der Reihenfolge hinzufügen

Um die Reihenfolge für Datensätze festzulegen, benötigen wir natürlich ein entsprechendes Feld. Dieses nennen wir **Position** und fügen es einfach der betroffenen Tabelle, in diesem Fall **tblArtikel**, als Feld mit dem Datentyp **Zahl** hinzu (siehe Bild 1). Hier passen wir allerdings noch die Feldgröße auf **Single** an, damit wir auch Dezimalzahlen eingeben können.

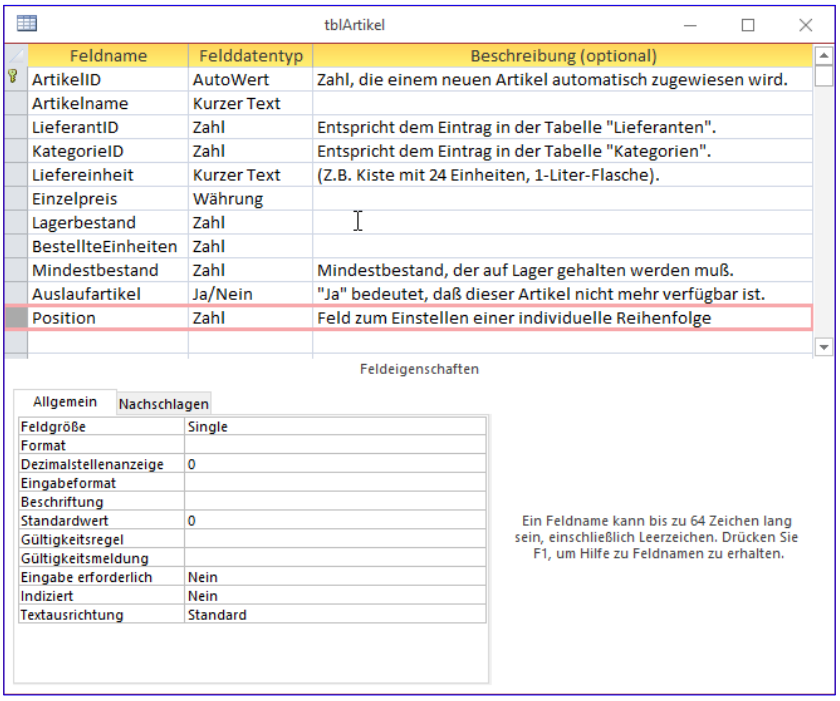


Bild 1: Feld für das Einstellen der Reihenfolge

Haupt- und Unterformular

Die Daten wollen wie in einem Unterformular in der Datenblattansicht anzeigen. Dazu erstellen wir zunächst ein neues Unterformular und fügen diesem eine Abfrage als Datenherkunft hinzu, welche alle Felder der Tabelle **tblArtikel** liefert, und zwar sortiert nach dem Feld **Position** (siehe Bild 2).

Danach können wir dem Unterformular die Felder der Datenherkunft zuweisen, allerdings in

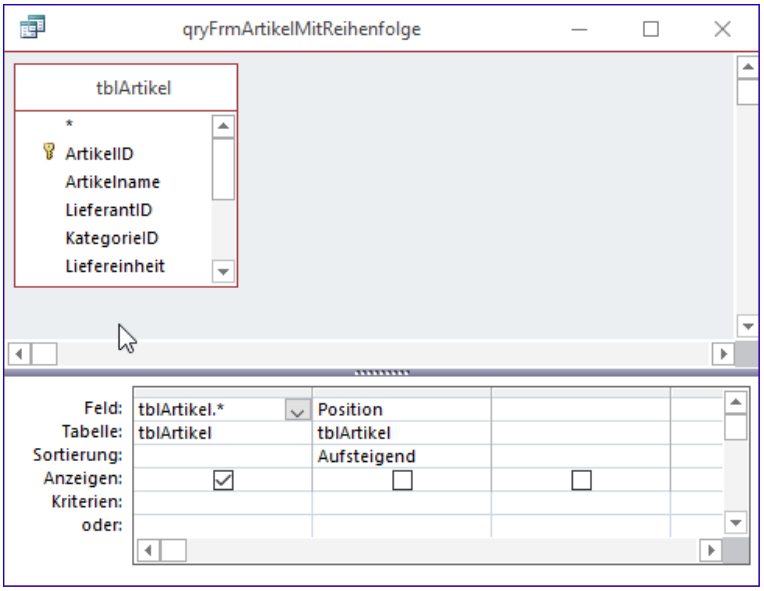


Bild 2: Datenherkunft für das Unterformular

einer ganz bestimmten Reihenfolge: Das Feld **Position** greifen wir zuerst aus der Feldliste heraus und fügen es oben im Formularentwurf ein.

Erst danach ziehen wir die übrigen Felder aus der Feldliste unterhalb des Feldes **Position** in das Zielformular (siehe Bild 3).

Dadurch werden die Felder in der Datenblattansicht gleich in der gewünschten Reihenfolge angezeigt, nämlich mit dem Feld **Position** an erster Stelle.

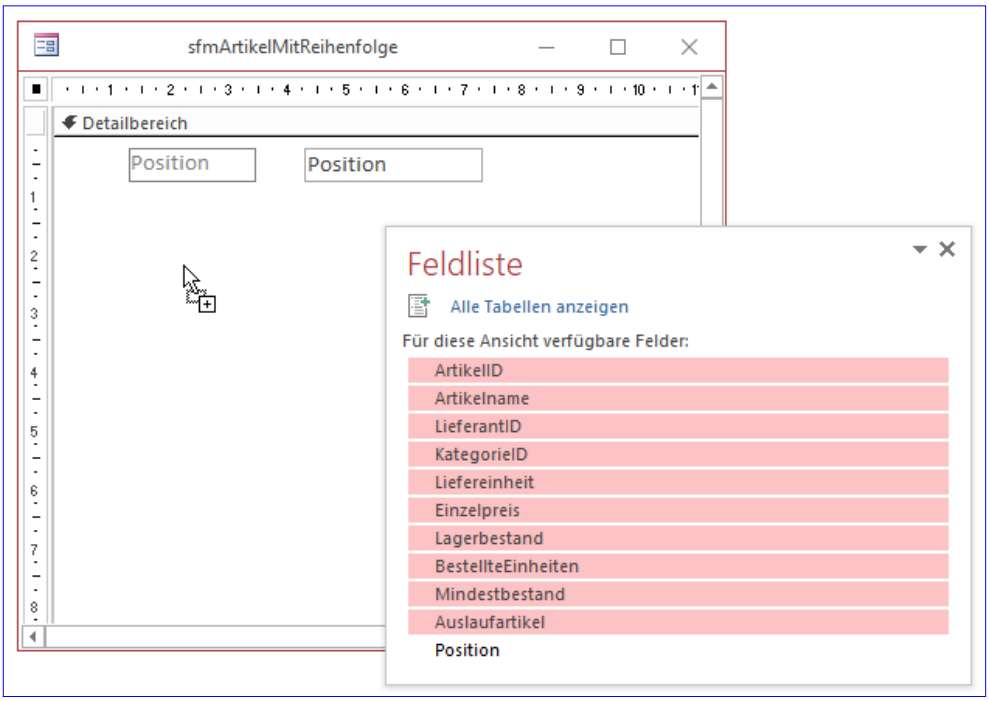


Bild 3: Hinzufügen der Felder zum Unterformular

Legen Sie nun noch die Eigenschaft **Standardansicht** auf den Wert **Datenblatt** fest und speichern Sie das Formular unter dem Namen **sfmArtikelMitReihenfolge**. Schließen Sie das Formular anschließend.

Danach erstellen Sie das Hauptformular und speichern es unter dem Namen **frmArtikelMitReihenfolge**. Ziehen Sie dann das Unterformular aus dem Navigationsbereich von Access in den Entwurf des neu angelegten Hauptformulars.

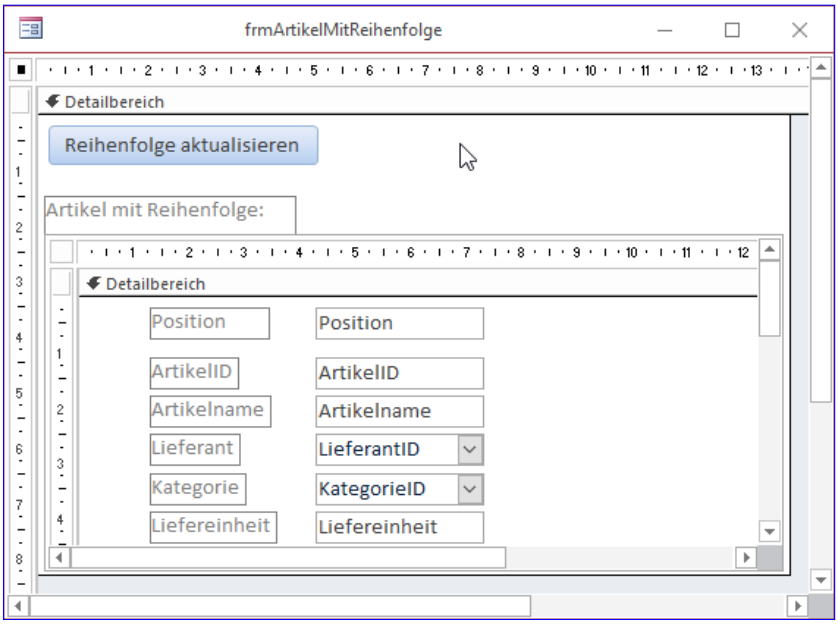


Bild 4: Unterformular zum Hauptformular hinzufügen

Richten Sie das Unterformular so aus, dass das Ergebnis wie in Bild 4 aussieht. Nun können Sie noch die Eigenschaften **Horizontaler Anker** und **Vertikaler Anker** des Unterformular-Steuerelements auf die Werte **Beide** einstellen, damit dieses seine Größe gemeinsam mit dem Hauptformular ändert.

Reihenfolge ohne Reihenfolge

Interessant ist, was geschieht, wenn Sie das Formular nun in der Formularansicht öffnen, ohne zuvor die Werte des Feldes **Position** zu füllen. Die Datensätze werden willkür-

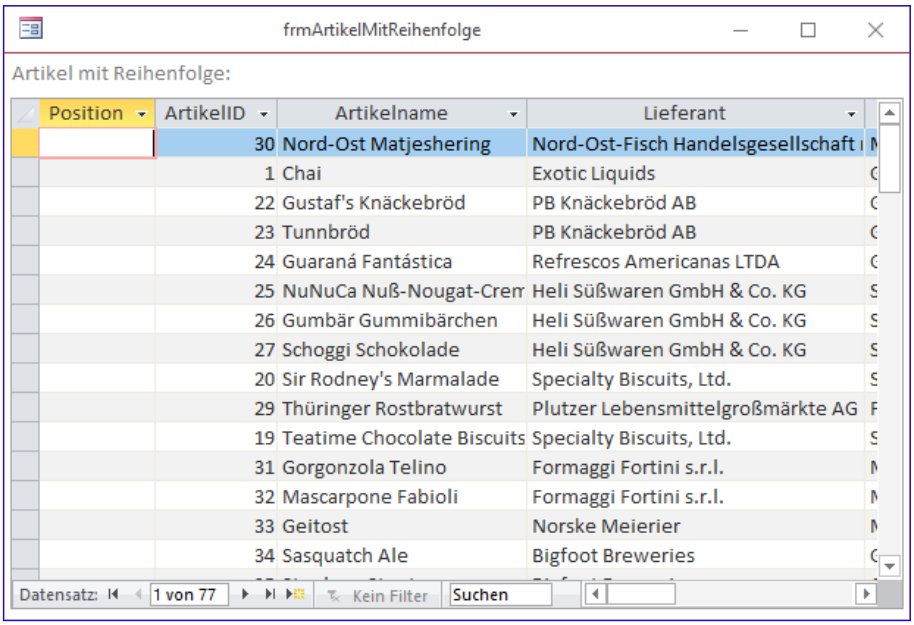


Bild 5: Ohne Werte liefert das Feld **Position** eine willkürliche Sortierung.

lich sortiert – und zwar noch nicht einmal nach dem Feld mit dem Primärindex (**ArtikelID**). Das Ergebnis sieht dann zunächst wie in Bild 5 aus.

Schaltfläche zum Aktualisieren der Reihenfolge

Nun fügen wir dem Hauptformular eine Schaltfläche hinzu, mit der wir die Reihenfolge der Datensätze aktualisieren können. Diese Schaltfläche nennen wir **cmdReihenfolgeAktualisieren**.

Für die das Ereignis **Beim Klicken** der Schaltfläche hinterlegen wir die Prozedur aus Listing 1. Diese Prozedur erstellt zunächst ein Recordset auf Basis der Tabelle **tblArtikel**, wobei diese die Datensätze aufsteigend nach den Feldern **Position** und **ArtikelID** sortiert. Danach durchläuft sie alle Datensätze der Tabelle in einer **Do While**-Schleife.

Hier versetzt sie den aktuellen Datensatz zunächst mit der **Edit**-Methode in den Bearbeitungszustand, stellt dann das Feld **Position** auf den Wert der Eigenschaft **AbsolutePosition** des Datensatzzeigers plus ein ein (**AbsolutePosition** ist 0-basiert) und speichert den Datensatz mit der **Update**-Methode. Danach werden die übrigen Datensätze in der **Do While**-Schleife auf die gleiche Weise bearbeitet. Die letzte Anweisung der Proze-

```
Private Sub cmdReihenfolgeAktualisieren_Click()
    Dim db As DAO.Database
    Dim rst As DAO.Recordset
    Set db = CurrentDb
    Set rst = db.OpenRecordset("SELECT * FROM tblArtikel ORDER BY Position, ArtikelID", dbOpenDynaset)
    Do While Not rst.EOF
        rst.Edit
        rst!Position = rst.AbsolutePosition + 1
        rst.Update
        rst.MoveNext
    Loop
    rst.Close
    Set rst = Nothing
    Set db = Nothing
    Me!sfmArtikelMitReihenfolge.Form.Requery
End Sub
```

Listing 1: Initiales Einstellen der Reihenfolge

dur aktualisiert die im Unterformular angezeigten Daten, die dann in der soeben festgelegten Sortierung erscheinen. Da das Feld **Position** zuvor noch keine Werte aufwies, erfolgt die Sortierung in diesem Fall ausschließlich nach den Werten des Primärschlüsselfeldes **ArtikelID**.

Benutzer gibt Reihenfolge vor

Nun soll der Benutzer eine einfache Möglichkeit erhalten, für einen neuen Datensatz direkt über das Feld **Position** an der gewünschten Stelle einzufügen. Nun ist es unter Access ja nicht möglich, neue Datensätze an beliebiger Stelle einzufügen, denn der neue, leere Datensatz ist immer der letzte ganz unten.

Durch unsere Vorbereitung kann der Benutzer jedoch nun einen neuen Datensatz eingeben und für diesen durch einen geeigneten Wert für das Feld **Position** nach dem Speichern an der richtigen Stelle einsortieren. Im Beispiel aus Bild 6 geben wir beispielsweise den Wert **1,5** für das Feld **Position** ein, weil wir den Datensatz zwischen dem ersten und zweiten vorhandenen Datensatz einsortieren wollen.

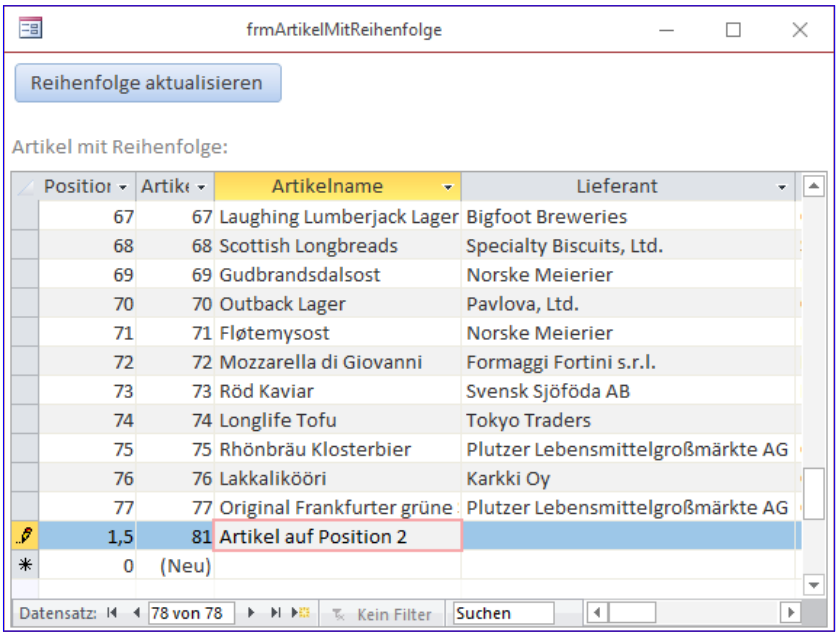


Bild 6: Der neue Datensatz mit Position 1,5 ...

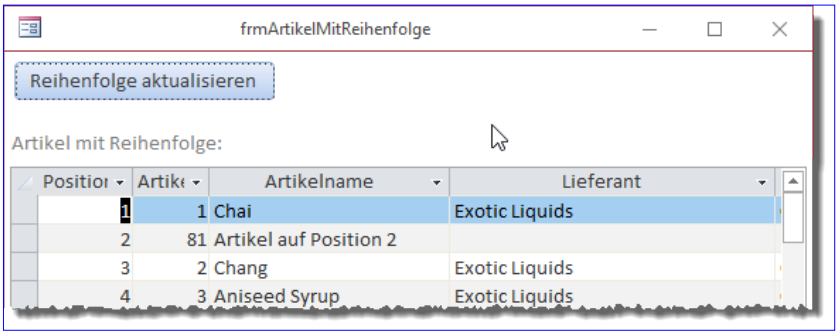


Bild 7: ... landet zwischen dem vorherigen ersten und zweiten Datensatz.

Nach dem Speichern des Datensatzes und dem anschließenden Betätigen der Schaltfläche **Reihenfolge aktualisieren** landet der Datensatz tatsächlich an der gewünschten Stelle (siehe Bild 7).

Reihenfolge ohne Schaltfläche

Nun wollen wir diese Lösung noch so verfeinern, dass der Benutzer den Datensatz nur noch speichern muss und der Datensatz dann automatisch an die gewünschte Stelle verschoben wird. Dazu fügen wir die Anweisungen, die sich bisher in der Prozedur **cmdReihenfolgeAktualisieren_Click** befanden, in der Ereignisprozedur ein, die durch das Ereignis **Nach Aktualisierung** des Unterformulars ausgelöst wird.

Die Ereignisprozedur sieht dann wie in Listing 2 aus. Sie führt grundsätzlich die gleichen Schritte durch wie die zuvor beschriebene Ereignisprozedur – mit dem Unterschied, dass sie gleich nach der Eingabe des neuen Datensatzes ausgeführt wird.

Und es gibt noch einen weiteren Einsatzzweck: Sie können nämlich auch die Position vorhandener Datensätze durch Einstellen des Wertes für das Feld **Position** ändern. Wenn Sie etwa den Datensatz an der zweiten Position auf

Löschen im Listenfeld per Tastatur

Wenn Sie Daten im Listenfeld anzeigen, die durch den Benutzer etwa per Doppelklick bearbeitet oder per Schaltfläche gelöscht oder erweitert werden sollen, lässt sich dies leicht erledigen. Eher selten trifft man auf Listenfelder, deren Einträge man einfach durch Markieren und Betätigen der Entf-Taste löschen kann. Wie Sie dies programmieren, schauen wir uns im vorliegenden Beitrag an.

Einfach Daten im Listenfeld markieren und mit der Entf-Taste löschen – das wäre in vielen Fällen eine praktische Erweiterung der sonst üblichen Löschen-Schaltfläche. Wenn man mehr als einen Datensatz aus der Liste löschen möchte, muss man sonst nämlich ordentlich mit der Maus arbeiten – markieren, auf Löschen klicken, markieren, auf Löschen klicken ...

Das können wir auch einfacher programmieren und dem Benutzer so die Anwendung vereinfachen.

Einträge aus ungebundenem Listenfeld löschen

Als Erstes schauen wir uns an, wie wir Einträge aus einem ungebundenen Listenfeld löschen können. Dieses legen wir unter dem Namen **lstUngebunden** in einem neuen Formular an. Das Formular soll beim Laden einige Einträge zum Listenfeld hinzufügen, was wir mit folgender Ereignisprozedur erledigen:

```
Private Sub Form_Load()
    Dim i As Integer
    Me!lstUngebunden.RowSourceType = "Value List"
    For i = 1 To 10
        Me!lstUngebunden.AddItem "Eintrag " & i
    Next i
End Sub
```

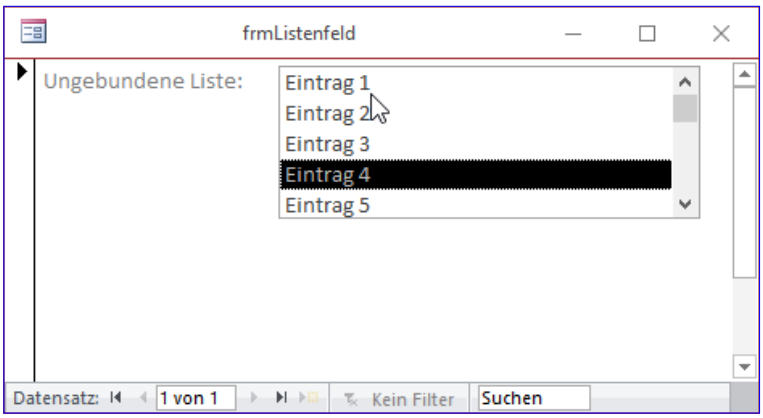


Bild 1: Listenfeld mit Beispieldaten

Die Prozedur stellt die Eigenschaft **Herkunftsart** des Listenfeldes auf **Wertliste** ein und fügt innerhalb einer **For...Next**-Schleife zehn durchnummerierte Einträge zum Listenfeld hinzu.

Dies liefert das Listenfeld mit den Einträgen aus Bild 1. Wenn wir nun einen der Einträge per Tastatur mit der **Entf**-Taste löschen wollen, müssen wir die Tastenereignisse des Listenfeldes hinzuziehen. In diesem Fall wollen wir den Eintrag löschen, sobald der Benutzer die **Entf**-Taste herunterdrückt. Dazu hinterlegen wir die Ereignisprozedur aus Listing 1 für das Ereignis **Bei Taste ab** des Listenfeldes.

Die Prozedur prüft, ob der Wert des Parameters **KeyCode**, der beim Herunterdrücken der Taste geliefert wird, der Zahl **46** entspricht. In diesem Fall ermittelt die Prozedur den Index des aktuell markierten Eintrags im Listenfeld und schreibt diesen in die Variable **lngItem**.

Diese Variable prüft die folgende **If...Then**-Bedingung auf den Wert **-1**. Ist der Wert nicht gleich **-1**, ruft die Prozedur die **RemoveItem**-Methode des Listenfeldes auf und übergibt mit **lngItem** den Index der zu löschenden Zeile. Damit wird der aktuell markierte Eintrag

```
Private Sub lstUngebunden_KeyDown(KeyCode As Integer, Shift As Integer)
    Dim lngItem As Long
    Select Case KeyCode
        Case 46
            lngItem = Me!lstUngebunden.ListIndex
            If Not lngItem = -1 Then
                Me!lstUngebunden.RemoveItem lngItem
                If Not Me!lstUngebunden.ListCount = 0 Then
                    If lngItem < Me!lstUngebunden.ListCount Then
                        Me!lstUngebunden.ListIndex = lngItem
                    Else
                        Me!lstUngebunden.ListIndex = Me!lstUngebunden.ListCount - 1
                    End If
                End If
            End If
    End Select
End Sub
```

Listing 1: Löschen einzelner Einträge aus einem ungebundenen Listenfeld

Damit kann der Benutzer nun durch mehrmaliges Betätigen der **Entf**-Taste mehrere untereinander liegende Einträge hintereinander löschen, ohne zwischendurch einen neuen Datensatz markieren zu müssen. Soll ein Datensatz nicht gelöscht werden, kann der Benutzer den nächsten Datensatz mit den **Nach oben**- und **Nach unten**-Tasten ansteuern.

entfernt. Das Listenfeld behält den Fokus, aber danach ist kein Datensatz markiert (siehe Bild 2). Sollten wir die Markierung nach dem Löschen auf einen anderen Datensatz verschieben? Denkbar wäre, dass der Benutzer beispielsweise von oben nach unten einige Datensätze löschen möchte. Also markieren wir nach dem Löschen den Datensatz, der sich direkt unterhalb des gelöschten Datensatzes befunden hat.

Dazu haben wir hinter der Zeile mit der **RemoveItem**-Methode zunächst die folgende Zeile eingefügt:

```
Me!lstUngebunden.ListIndex = lngItem
```

Allerdings war dies etwas zu kurz gedacht, denn die Prozedur lieferte einen Fehler, wenn wir den letzten vorhandenen Eintrag gelöscht haben – und auch das Löschen des letzten Eintrags der Liste lieferte regelmäßig einen Fehler.

Also haben wir noch ein paar verschachtelte Bedingungen hinzugefügt. Die erste fragt ab, ob das Listenfeld überhaupt noch Einträge enthält. Falls nicht, wird auch kein Eintrag mehr markiert. Falls doch, prüfen wir, ob **lngItem**, also der Index des gelöschten und des nun zu markierenden Datensatzes, der letzte Eintrag im Listenfeld ist. Falls nein, wird einfach der Eintrag mit dem Index des gelösch-

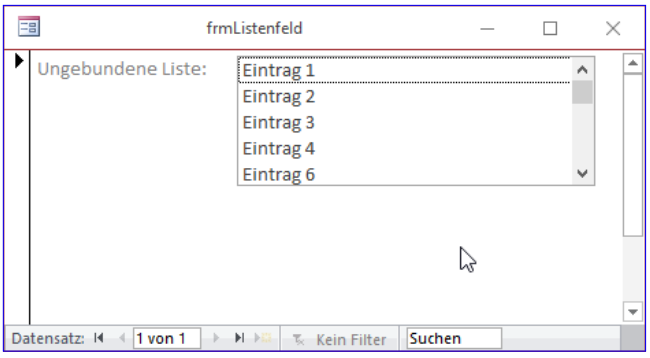


Bild 2: Listenfeld nach dem Löschen eines Datensatzes

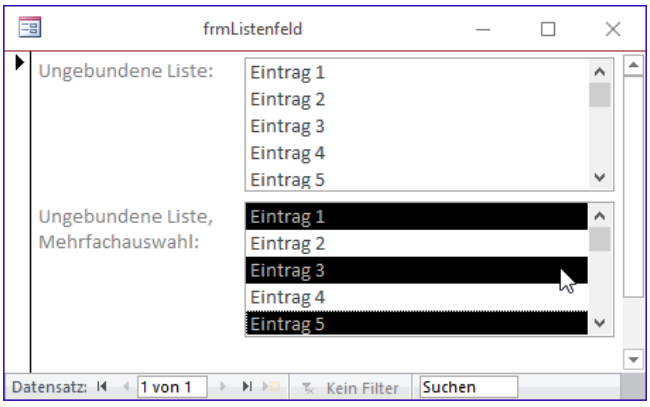


Bild 3: Löschen einer Mehrfachauswahl

ten Eintrags markiert, also der erste Eintrag hinter dem gelöschten Eintrag. Sollten wir uns schon an der letzten Position befinden, markiert die Prozedur den letzten Eintrag des Listenfeldes.

Auf diese Weise können wir auch den hinteren Eintrag markieren und von dort aus alle Einträge von hinten nach vorne löschen, ohne dass wir zwischendurch einen Eintrag von Hand markieren müssen.

```
Private Sub lstUngebundenMehrfach_KeyDown(KeyCode As Integer, Shift As Integer)
    Dim varItem As Variant
    Select Case KeyCode
        Case 46
            For Each varItem In Me!lstUngebundenMehrfach.ItemsSelected
                Me!lstUngebundenMehrfach.RemoveItem varItem
            Next varItem
    End Select
End Sub
```

Listing 2: Erster Versuch, Einträge aus der Mehrfachauswahl eines Listenfeldes zu löschen

```
Private Sub lstUngebundenMehrfach_KeyDown(KeyCode As Integer, Shift As Integer)
    Dim varItem As Variant
    Dim arrItems() As Long
    Dim l As Long
    Dim i As Integer
    Select Case KeyCode
        Case 46
            For Each varItem In Me!lstUngebundenMehrfach.ItemsSelected
                ReDim Preserve arrItems(i)
                arrItems(i) = varItem
                i = i + 1
            Next varItem
            For l = i - 1 To 0 Step -1
                Me!lstUngebundenMehrfach.RemoveItem arrItems(l)
            Next l
    End Select
End Sub
```

Listing 3: Zweiter Versuch, Einträge aus der Mehrfachauswahl eines Listenfeldes zu löschen

Mehrfachauswahl löschen

Nun wollen wir uns ansehen, wie wir Einträge aus einem Listenfeld löschen können, für das die Mehrfachauswahl aktiviert ist. Dazu fügen wir dem Beispielformular ein weiteres Listenfeld namens **lstUngebundenMehrfach** hinzu, dessen Eigenschaft **Mehrfachauswahl** wir auf **Einfach** oder **Erweitert** einstellen. Damit auch dieses Listenfeld mit einigen Beispieleinträgen gefüllt wird, fügen wir der **For...Next**-Schleife der Prozedur **Form_Load** die folgende Zeile hinzu:

```
Me!lstUngebundenMehrfach.AddItem "Eintrag " & i
```

In diesem Fall können wir nicht nur einen, sondern mehrere Einträge markieren (siehe Bild 3). Wir fügen der Prozedur, die durch das Ereignis **Bei Taste ab** des Listenfeldes **lstUngebundenMehrfach** ausgelöst wird, die Zeilen aus Listing 2 hinzu.

Wenn wir dann die Einträge wie im Bild oben markieren und auf die **Entf**-Taste klicken, wird nur der erste Eintrag gelöscht. Warum das? Weil durch das Löschen des ersten Eintrags und damit der Aktualisierung des Listenfeldes auch die vorhandenen Markierungen entfernt werden.

Also müssen wir dies etwas anders angehen und die Indizes der zu löschenden Einträge erst in einem Array zwischenspeichern.

Die programmieren wir in der neuen Version der Prozedur, die Sie in Listing 3 finden. Hier verwenden wir gleich drei neue Variablen, nämlich **i** und **l** mit dem Datentyp **Long**

Zeiträume per Listenfeld und InputBox

Die Beispieldatenbank Vereinsverwaltung erwartet an einer Stelle die Eingabe der Vereinszugehörigkeit. Je Mitglied kann es auch mehrere Vereinszugehörigkeiten geben. Diese wollen wir diesmal in einem Listenfeld darstellen und ohne Verwendung eines eigenen Detailformulars per InputBox verwalten. Dass dies gar nicht unbedingt einfacher oder schneller geht, zeigt der vorliegende Beitrag.

Der größte Aufwand bei der Verwaltung von Zeiträumen entsteht durch die Überprüfung der eingegebenen Werte: Das Startdatum darf nicht hinter dem Enddatum liegen, die Zeiträume der verschiedenen Datensätze dürfen sich nicht überlappen. In diesem Beitrag zeigen wir, wie Sie die Zeiträume zu einem bestimmten Thema, hier die Vereinszugehörigkeiten von Mitgliedern zu einem Verein, in einem Listenfeld anzeigen und per **InputBox** verwalten wollen. Diese Zeiträume könnten aber auch etwa die Beschäftigungszeiten in einem Unternehmen sein – und Sie finden sicher noch weitere Beispiele. In unserem Beispiel kümmern wir uns um die Einträge der Tabelle **tblVereinszugehoerigkeiten**, die wie in Bild 1 mit der Tabelle **tblMitglieder** verknüpft ist.

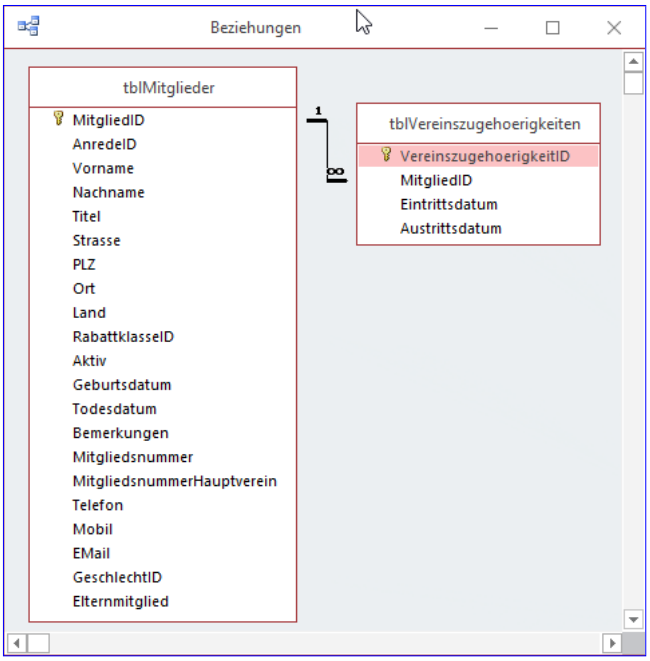


Bild 1: Mitglieder und Vereinszugehörigkeiten

Das Listenfeld **lstVereinszugehoerigkeiten** sowie die beiden Schaltflächen **cmdNeueVereinszugehoerigkeit** und **cmdVereinszugehoerigkeitLoeschen** füllen wir gleich mit Code. Ein Klick auf die Schaltfläche **cmdNeueVereinszugehoerigkeit** soll beispielsweise eine **InputBox** wie in Bild 2 anzeigen.

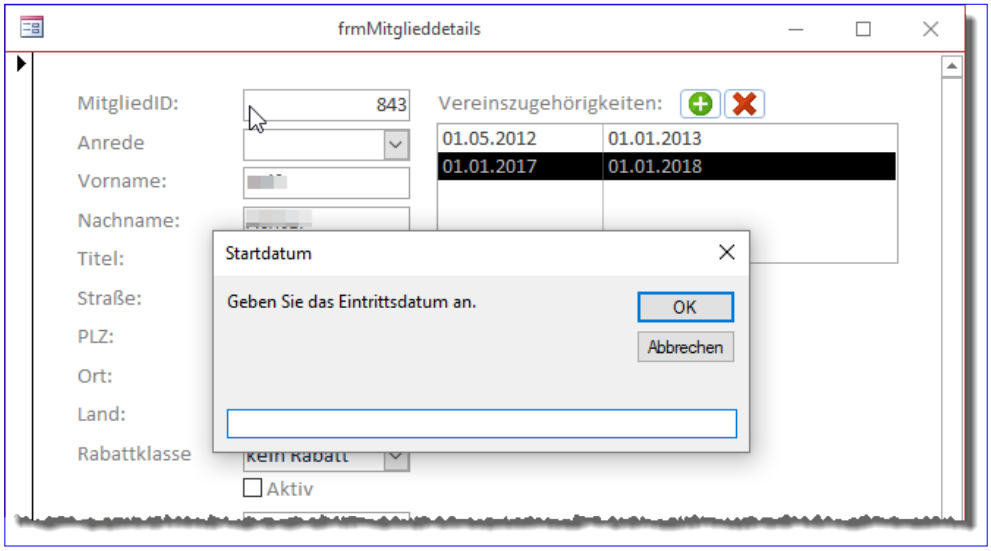


Bild 2: Zeiträume per InputBox eingeben

Zuvor wollen wir jedoch noch die Daten der Tabelle **tblVereinszugehoerigkeiten** im Listenfeld anzeigen. Wie Sie diese Steuerelemente anlegen, erläutern wir im Beitrag

Vereinsverwaltung: Formulare (**www.access-im-unternehmen.de/1134**).

Datensatzherkunft für das Listenfeld

Das Listenfeld soll alle Datensätze der Tabelle **tblVereinszugehoerigkeiten** anzeigen, die dem aktuell im Formular **frmMitglieddetails** angezeigten Mitglied zugeordnet sind. Außerdem sollen die Daten aufsteigend nach dem Eintrittsdatum sortiert werden. Dies erledigen wir mit der Abfrage aus Bild 3. Diese verwendet alle Felder der Tabelle **tblVereinszugehoerigkeiten**, das Fremdschlüsselfeld **MitgliedID** wird jedoch nur als Kriterium verwendet. Damit nur die Vereinszugehörigkeiten zum aktuellen Mitglied erscheinen, stellen wir das Kriterium dieses Feldes auf **[Forms]![frmMitglieddetails]![MitgliedID]** ein. Dies führt dazu, dass die Datensätze automatisch gefiltert werden. Damit das Listenfeld nur die für die Bearbeitung wichtigen Spalten anzeigt, stellen wir die Eigenschaft **Spaltenanzahl** auf **3** und die Eigenschaft **Spaltenbreiten** auf **0cm;3cm;3cm** ein.

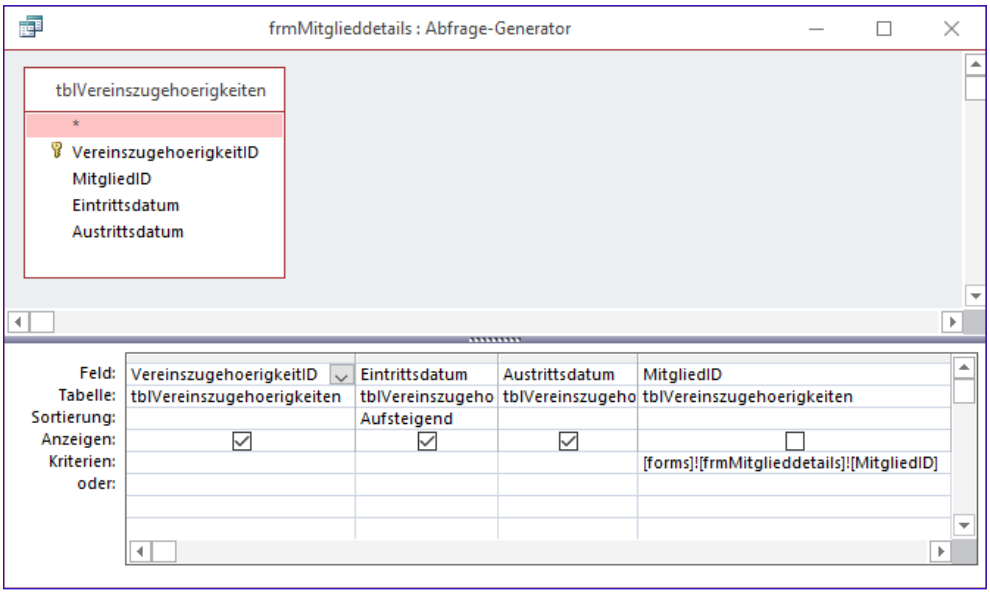


Bild 3: Datensatzherkunft des Listenfeldes **lstVereinszugehoerigkeiten**

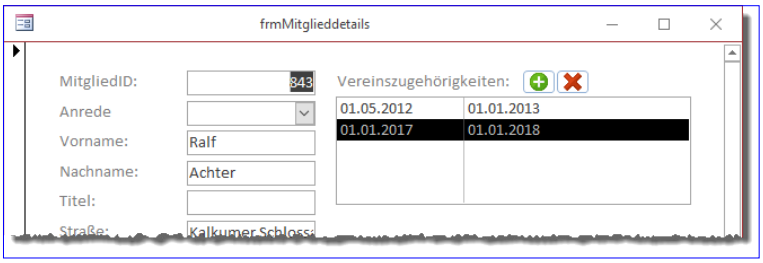


Bild 4: Zwei Datensätze im Listenfeld

Damit erhalten wir dann, sofern schon einige Datensätze vorliegen, in der Formularansicht Daten wie etwa in Bild 4. Wenn wir dann den Datensatz im Formular wechseln, aktualisiert dies allerdings noch nicht die im Listenfeld angezeigten Daten. Dazu müssen wir noch die Ereignisprozedur **Beim Anzeigen** zum Formular hinzufügen und diese wie folgt ergänzen:

```
Private Sub Form_Current()
    Me!lstVereinszugehoerigkeiten.Requery
End Sub
```

Danach gelingt die Aktualisierung des Listenfeldes beim Datensatzwechsel reibungslos.

Hinzufügen eines neuen Zeitraums

Die durch die Schaltfläche **cmdNeueVereinszugehoerigkeit** ausgelöst Ereignisprozedur soll dem Benutzer die Möglichkeit geben, einen neuen Zeitraum zum aktuellen Mitglied hinzuzufügen (siehe Listing 1). Die Prozedur ermittelt zunächst mit einer **InputBox**-Anweisung das Eintrittsdatum für die neue Vereinszugehörigkeit. Dann prüft sie, ob die Variable, welche das Ergebnis der Funktion aufgenommen hat, eine Zeichenkette mit einer Länge größer **0** aufweist. Dies ist unter folgenden Bedingungen nicht der Fall: Wenn der Benutzer das Feld leer lässt und auf **OK** klickt oder wenn er direkt auf **Abbrechen** klickt – dann wird unabhängig vom aktuell in die **InputBox** eingetragenen Wert eine leere Zeichenfolge zurück-

geliefert. Sollte **strEintritt** nach dieser Zeile also eine Zeichenkette mit mindestens einem Zeichen enthalten, ist die folgende **If...Then**-Bedingung erfüllt. Eine weitere **If...Then**-Bedingung prüft dann, ob der Benutzer ein gültiges Datum eingegeben hat. Falls nicht, wird der **Else**-Teil ausgeführt, welcher den Benutzer darauf hinweist, dass er

```
Private Sub cmdNeueVereinszugehoerigkeit_Click()
    Dim db As DAO.Database
    Dim datEintritt As Date
    Dim datAustritt As Date
    Dim strEintritt As String
    Dim strAustritt As String
    strEintritt = InputBox("Geben Sie das Eintrittsdatum an.", "Startdatum")
    If Len(strEintritt) > 0 Then
        Set db = CurrentDb
        If IsDate(strEintritt) Then
            datEintritt = CDate(strEintritt)
            strAustritt = InputBox("Geben Sie das Austrittsdatum an.", "Austrittsdatum", "Noch aktiv.")
            If Len(strAustritt) > 0 And Not strAustritt = "Noch Aktiv." Then
                If IsDate(strAustritt) Then
                    datAustritt = CDate(strAustritt)
                    If datEintritt > datAustritt Then
                        MsgBox ("Das Eintrittsdatum muss vor dem Austrittsdatum liegen. Die Eingabe wird abgebrochen.")
                        Exit Sub
                    Else
                        db.Execute "INSERT INTO tblVereinszugehoerigkeiten(MitgliedID, Eintrittsdatum, " _
                            & "Austrittsdatum) VALUES(" & Me!MitgliedID & ", " & SQLDatum(datEintritt) & ", " _
                            & SQLDatum(datAustritt) & ")", dbFailOnError
                        Me!lstVereinszugehoerigkeiten.Requery
                    End If
                Else
                    MsgBox ("'" & strAustritt & "' ist kein gültiges Austrittsdatum.")
                    Exit Sub
                End If
            Else
                db.Execute "INSERT INTO tblVereinszugehoerigkeiten(MitgliedID, Eintrittsdatum) VALUES(" _
                    & Me!MitgliedID & ", " & SQLDatum(datEintritt) & ")", dbFailOnError
                Me!lstVereinszugehoerigkeiten.Requery
            End If
        Else
            MsgBox ("'" & strEintritt & "' ist kein gültiges Eintrittsdatum.")
            Exit Sub
        End If
    Else
        Exit Sub
    End If
End Sub
```

Listing 1: Anlegen eines neuen Zeitraums

kein gültiges Eintrittsdatum eingegeben hat. Ist das Datum jedoch gültig, speichern wir die in den Datentyp **Date** konvertierte Zeichenkette in der Variablen **datEintritt**. Danach können wir mit der Abfrage des Austrittsdatums fortfahren. Normalerweise gehen wir davon aus, dass eine neue Vereinszugehörigkeit nur ein Eintrittsdatum enthält, daher füllen wir den Standardtext dieser **InputBox** mit dem Text **Noch aktiv.**, was wie in Bild 5 aussieht.

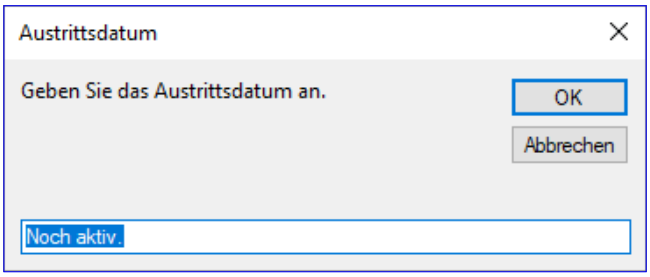


Bild 5: Vorbelegung für das Austrittsdatum

Nach der Eingabe des Benutzers prüft die folgende **If... Then**-Bedingung, ob sowohl die zurückgelieferte Zeichenkette eine Länge von mindestens einem Zeichen aufweist und der Text nicht mit dem Standardwert **Noch aktiv.** übereinstimmt. Falls nicht, wird im **Else**-Teil der Bedingung über die **Execute**-Methode des **Database**-Objekts **db** eine **INSERT INTO**-Anweisung ausgeführt, welche einen neuen Datensatz in der Tabelle **tblVereinszugehoerigkeiten** anlegt und dabei die Felder **MitgliedID** und **Eintrittsdatum** mit dem Primärschlüsselwert des aktuellen Datensatzes im Formular und dem Wert aus **datEintritt** füllt. Dabei nutzen wir die Hilfsfunktion **SQLDatum** aus dem Modul **mdlTools**, um das Datum SQL-konform abzubilden (als **yyyy-mm-dd**). Danach aktualisieren wir die Anzeige im Listenfeld mit der **Requery**-Methode des Listenfeldes.

Hat der Benutzer jedoch ein gültiges Austrittsdatum angegeben, weil er etwa aus statistischen Gründen frühere Vereinszugehörigkeiten nachtragen möchte, prüft die Prozedur, ob es sich bei **strAustritt** um ein gültiges Datum handelt. Auch hier erscheint im **Else**-Teil eine entsprechende Meldung, falls **strAustritt** kein gültiges Datum enthält.

Ist **strAustritt** ein Datum, wird es mit **CDate** in die

Date-Variable **datAustritt** überführt. Da nun beide Datumsangaben vorhanden sind, können wir noch prüfen, ob das Eintrittsdatum gegebenenfalls hinter dem Austrittsdatum liegt und den Benutzer darauf hinweisen. Er muss den Vorgang dann erneut starten. Sind die Daten hingegen korrekt, fügt die Prozedur den Datensatz mit einer **INSERT INTO**-Aktionsabfrage zur Tabelle **tblVereinszugehoerigkeiten** hinzu. Diesmal berücksichtigt sie dabei nicht nur die Felder **MitgliedID** und **Eintrittsdatum**, sondern auch noch das Feld **Austrittsdatum**.

Ändern eines bestehenden Zeitraums

Wenn der Benutzer doppelt auf einen der vorhandenen Einträge klickt, löst dies die Ereignisprozedur **lstVereinszugehoerigkeiten_DblClick** aus, die Sie in Listing 2 finden.

Die Prozedur ermittelt zunächst über die Eigenschaft **Column(0)** des Listenfeld-Steuerelements den Wert des in der ersten Spalte angezeigten Feldes. Wobei »angezeigt« der falsche Begriff ist, denn wir haben die Breite dieser Spalte ja auf **0cm** eingestellt, wodurch die Spalte faktisch ausgeblendet wird. Der darin enthaltene Wert des Primärschlüsselfeldes der Tabelle **tblVereinszugehoerigkeiten** für den aktuell markierten Datensatz landet jedenfalls in der Variablen **lngVereinszugehoerigkeitID**. Gegebenenfalls hat der Benutzer auf eine leere Stelle im Listenfeld geklickt. In diesem Fall liefert **Column(0)** den Wert **Null** zurück, den wir durch die Funktion **Nz** durch den Wert **0** ersetzen. Hat **lngVereinszugehoerigkeitID** danach den Wert **0**, zeigt die Prozedur eine Meldung an, die den Benutzer darauf hinweist, dass er den Doppelklick für einen bestehenden Eintrag ausführen muss. Nun folgt ein vorbereitender Schritt für die Bearbeitung der bestehenden Daten: Wir speichern die in der zweiten und dritten Spalte des markierten Eintrags im Listenfeld befindlichen Werte in den Variablen **strEintritt** und **strAustritt**. Dann

SVG als Grafikkomponente

In zwei Beiträgen rund um das Thema 'HTML5 als Grafikkomponente' erfuhren Sie, wie Sie mithilfe des Webbrowser-Steuerelements und HTML5 programmgesteuert Grafiken erzeugen. Über eine andere Schnittstelle, nämlich SVG, lässt sich dasselbe erreichen, wobei hier nicht Pixel-, sondern Vektorgrafiken erzeugt werden, die deutlich mehr Interaktivität zulassen und nachträglich bearbeitbar sind.

Referenz

Die Ausführungen dieses Beitrags setzen jene aus dem Artikel **HTML5 als Grafikkomponente** in der Ausgabe **01/2018** fort. Dort sind die Grundlagen zum Einsatz des **Webbrowser-Steuerelements** im Verein mit **HTML5** beschrieben, auf die hier deshalb nicht noch einmal eingegangen wird.

SVG als Alternative zu HTML5

Über **HTML5** können Sie unter Access im Webbrowser-Steuerelement Bilder anlegen, indem Sie dessen zugrundeliegendes **HTML**-Objektmodell steuern. Als Ergebnis erhalten Sie eine Pixelgrafik als Summe der Grafikoperationen, deren Schritte sich nicht mehr rückgängig machen oder bearbeiten lassen.

Lediglich das Abspeichern als Bilddatei oder im **BLOB**-Feld einer Tabelle steht Ihnen dann noch offen. Ganz anders bei **SVG**, wo die Grafik im Vektorformat entsteht, auf deren einzelne Elemente jederzeit zugegriffen werden kann. Solche Grafiken rendert der Browser dann zur Laufzeit.

Es verwundert eigentlich, dass diese flexible Lösung noch von niemand aufgegriffen und umgesetzt wurde – dachte ich! Tatsächlich findet, während ich dies zu Papier bringe, in Wien die zweite **Access-DevCon** statt, eine internationale Entwicklerkonferenz, welche unter anderem Vorträge von Experten zu verschiedenen Themenstellungen rund um Access anbietet. Und der litauische Entwickler **Alexandru Tofan**, ein bisher weitgehend unbeschriebenes Blatt, demonstriert dort ebenfalls seine Lösungen zum Einsatz von **SVG** als Grafik-Engine. Ich bin gespannt, welche Ähnlichkeit seine Ausführungen zu den hier ganz unabhängig vorgestellten haben werden, falls sie veröffentlicht werden...

Der Hauptgrund dafür, dass eine Implementierung von **SVG** im **Webbrowser Control** noch nicht stattfand, ist wohl darin zu suchen, dass Microsoft dem **Internet Explorer** erst spät den Umgang mit diesem Format beibrachte. Und da das Webbrowser-Steuerelement im Rohzustand nur die Version **7** des Browsers emuliert, welche noch kein **SVG** kannte, laufen die sehr wohl im **HTML**-Objektmodell enthaltenen zugehörigen Methodenaufrufe ins Leere. Erst die Modifikation des Webbrowsers über die VBA-Routine **PrepareBrowser**, die, wie im oben angeführtem Beitrag, schon mehrmals erläutert wurde, lässt ihn im Gewand des **IE 11** erscheinen. Danach akzeptiert er anstandslos auch alle **SVG**-Methoden.

SVG als Format

SVG (**Scalable Vector Graphics**) ist an sich ein auf **XML** basierendes Dateiformat, welches seit dem Jahr 2001 existiert und kontinuierlich weiterentwickelt wurde. Dateien mit der Endung **.svg** können etwa mit dem Open Source-Programm **Inkscape** angezeigt oder erstellt werden, welches sich fast schon zum Standard gemausert hat. Aber auch andere Vektorzeichenprogramme beherrschen meist den Export in dieses Format.

Inzwischen hat es sich aber vor allem im Web als wichtigste Schnittstelle zur Anzeige von Vektorgrafiken etabliert. Der Vorteil von Vektorgrafiken gegenüber Pixelbildern ist ja die verlustfreie Skalierbarkeit, denn solche Grafiken werden grundsätzlich erst zur Laufzeit beim

Rendern in Pixel übersetzt. Bei technischen Zeichnungen, Organigrammen, Charts oder Karten ist es deshalb das Format der Wahl.

Einst ging es in erster Linie um die Anzeige solcher **SVG**-Grafikdateien, für die bis dahin nur ein spezielles in die Website eingebettetes **Adobe**-ActiveX-Steuerelement infrage kam. Doch seit der Implementierung direkt im Browser kam auch die dynamische Steuerung der einzelnen Elemente der Grafik hinzu.

So lassen sich **SVG**-Grafiken nun auch ohne jeglichen Bezug zu einer Datei etwa über **Javascript** anlegen und beeinflussen. Und glücklicherweise hat Microsoft im **HTML**-Objektmodell des Internet Explorers alle Elemente von **Javascript** nach **COM** übersetzt, womit auch unter VBA Zugriff auf die entsprechenden Klassen und Methoden besteht, ohne dass Sie sich mit einer anderen Skriptsprache beschäftigen müssen.

Die **SVG**-Grafiken stellen nun einfach eine Folge von Grafikanweisungen dar, die im **XML**-Format abgelegt sind. Der Browser interpretiert diese und rendert sie anhand der übergebenen Parameter. Dafür gibt es das neue **Tag svg**, welches allerdings nur dann korrekt interpretiert wird, wenn ein bestimmtes **Namespace**-Attribut mit **xmlns** hinzugefügt wird:

```
<svg xmlns="http://www.w3.org/2000/svg"> (Inhalt) <svg>
```

Innerhalb dieses **Tags** befinden sich dann die Anweisungen. Das können Linien sein, Rechtecke, Kreise, Ellipsen, Polygone, Bezier-Kurven, Text, Füllungen, Pixeldaten oder Verweise auf Bilddateien. All diese Elemente können über Attribute mit diversen Parametern versehen werden, die deren Gestalt bestimmen. Der Clou aber sind mehrere Grafikfilter, die Sie auf einfache Weise anlegen und nach Belieben auf diese Elemente anwenden können. So kann etwa gefüllter Kreis mit einem **Blur**-Filter verwaschen werden, dessen Parameter Sie ebenfalls steuern können.

Eine einfache Line erzeugen Sie etwa über dieses Konstrukt:

```
<svg xmlns="http://www.w3.org/2000/svg"
    height="300" width="500">
<line x1="0" y1="0" x2="200" y2="200"
    style="stroke:black;stroke-width:2"/>
</svg>
```

Hier sind dem **svg**-Tag gleich die Attribute **width** und **height** zugeordnet, die die Abmessungen der Grafikfläche (**500x300** Pixel) festlegen. Die Linie erzeugen Sie über das **line**-Tag. Die Koordinatenpunkte der Linie sind über die Werte der Attributparameter **x1**, **y1**, **x2**, **y2** definiert. Das Aussehen der Linie bestimmt nun das **style**-Attribut, welches seinerseits eine Menge weiterer Definitionen aufnehmen kann, die sich aus dem Fundus der **HTML-Styles** ableiten, die Sie eventuell auch abseits von **SVG** kennen. So gibt **stroke** etwa an, welche Farbe die Line haben soll, was hier auf Schwarz (**black**) festgelegt ist.

Natürlich sind hier auch beliebige andere über Hexadezimalwerte definierte Farben möglich, wie **#A0A0A0**, oder über die **rgb**-Funktion ermittelte:

```
stroke:rgb(40,20,105)
```

Die Breite der Linie steht schließlich im Parameter **stroke-width** (2 Pixel).

Übrigens können Sie innerhalb des **svg-Tags** wiederum ein oder mehrere **svg-Tags** einbetten, was zu verschachtelten Zeichenflächen führt.

SVG dynamisch unter VBA erzeugen

Der Witz am **HTML-Objektmodell** ist aber ja, dass Sie den HTML-Text nicht unbedingt per String-Verkettung generieren müssen, sondern sich der passenden HTML-Klassen bedienen, deren Eigenschaften Sie dann einstellen. So gibt es zur Linie etwa das Element **SVGLineElement**, welches Sie so erzeugen:

```
Dim oDoc As HTMLDocument
Dim oSVGLine As SVGLineElement

Set oDoc = WebCtl.Object.Document
Set oSVGLine = oDoc.createElementNS( _
               "http://www.w3.org/2000/svg",
"line")
```

Das HTML-Dokument **oDoc** erhalten Sie hier über die **Document**-Eigenschaft des Webbrowser-Steuerelements **WebCtl**. Normalerweise reicht die Methode **createElement** aus, um auf der Website ein neues HTML-Element zu erzeugen. Hier allerdings wird die Alternative **createElementNS** benötigt, weil auf den **SVG-Namespace** Bezug genommen werden muss, damit der Webbrowser erkennt, um welches **line**-Element es sich genau handelt.

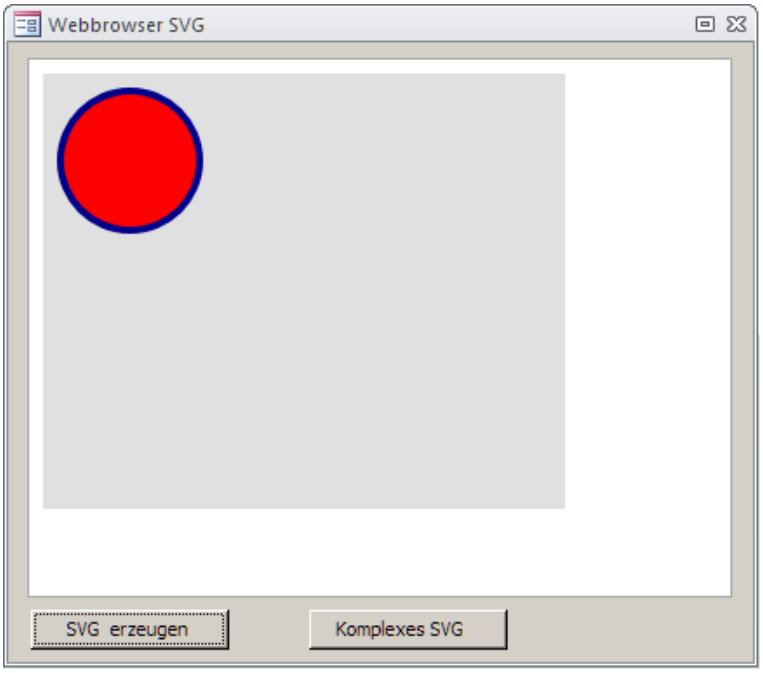


Bild 1: Eine im Webbrowser-Steuerelement per VBA generierte SVG-Grafik

So erzeugt stellen Sie nun die Eigenschaften des **SVG-LineElements** ein:

```
oSVGLine.x1 = 0: oSVGLine.y1 = 0
oSVGLine.x2 = 200: oSVGLine.y2 = 200
```

Leider sind das auch schon alle Eigenschaften, die diese Klasse bei Microsoft direkt anbietet. Alle anderen müssen Sie über die Methode **setAttribute** belegen:

```
oSVGLine.setAttribute "stroke", "black"
oSVGLine.setAttribute "stroke-width", "2"
```

Die Methode **setAttribute** finden Sie im Objektkatalog nicht unter der Klasse **SVGLineElement**, und trotzdem funktioniert sie. Das liegt daran, dass sich ein **SVG-LineElement** versteckt vom allgemeineren **IHTMLElement** ableitet, welches diese Methode aufweist.

SVG-Grafik-Beispiel

Das Formular **frmSVGTest** der Demodatenbank vereint die wesentlichen Techniken, die für den Einsatz von **SVG** als Grafikgenerator zum Zuge kommen. Bild 1 zeigt es zur Laufzeit, nachdem auf die Schaltfläche **SVG erzeugen** geklickt wurde.

Das Formular enthält lediglich ein Webbrowser-Steuerelement mit dem Namen **ctlIE**, dessen Steuerelementinhalt im Entwurf auf **"=about:blank"** festgelegt ist und einen Button, der die Anlage der **SVG**-Grafik auslöst. Damit das Webbrowser-Control mit **SVG** umgehen kann, muss zuvor die Routine **PrepareBrowser** des Moduls **mdlWebControl** aufgerufen worden sein, welche Einstellungen der Registry modifiziert, wie dies bereits erschöpfend in den Beiträgen zu **HTML5** erläutert wurde.

Der Klick auf den Button **cmdCreate** lässt die Ereignisprozedur in Listing 1 ablaufen. Hier kommt es zunächst zum Aufruf der Hilfsroutine **NavigateBlank**, die weiter unten abgebildet ist. Sie stellt per **Navigate2**-Methode sicher, dass im Webbrowser **ctlIE** ein leeres Dokument angelegt ist. Nachdem dieses erfolgreich fertiggestellt ist, was die **Do-Loop**-Schleife mit Abfrage der **ReadyState**-Eigenschaft kontrolliert, kann die modulweit deklarierte Objektvariable **oDoc** auf dieses Dokument gesetzt

werden. Das **WithEvents** in der Deklaration ermöglicht, dass auf Ereignisse der gesamten Website, wie Mausklicks, reagiert werden kann.

Im **Header** des Dokuments muss anschließend eine **meta**-Eigenschaft hinterlegt werden, die **X-UA-Compatible** lautet, damit der Browser die **SVG**-Anweisungen auch umsetzt. Dies geschieht durch Beschreiben des Tags über die Methode **innerHTML** des **head**-Objekts.

Und nun kann bereits die **SVG**-Zeichenfläche im Dokument mit **createElementNS** angelegt werden, wobei hier das Element-Tag **svg** als String übergeben wird. Das Element landet in der Objektvariablen **oSVG** vom Typ **SVGSVGElement**, die ebenfalls im Kopf des Moduls mit **WithEvents** deklariert ist, damit auf Ereignisse speziell der Zeichenfläche reagiert werden kann.

```
Private WithEvents oDoc As HTMLDocument
Private WithEvents oSVG As SVGSVGElement
Private oEvent As IHTMLEventObj

Private Sub cmdCreate_Click()
    Dim oSVGElement As SVGCircleElement
    Dim oElement As IHTMLElement

    NavigateBlank
    oDoc.head.innerHTML = "<head><meta http-equiv=""X-UA-Compatible""/></head>"

    Set oSVG = oDoc.createElementNS("http://www.w3.org/2000/svg", "svg")
    With oSVG
        .Width.baseVal.Value = 300
        .Height.baseVal.Value = 250
        .Style.backgroundColor = "#e0e0e0"
    End With

    Set oSVGElement = oDoc.createElementNS("http://www.w3.org/2000/svg", "circle")
    With oSVGElement
        .setAttribute "id", "Kreis"
        .setAttribute "cx", "50"
        .setAttribute "cy", "50"
        .setAttribute "r", "40"
        .setAttribute "stroke", "darkblue"
        .setAttribute "stroke-width", "4"
        .setAttribute "fill", "red"
    End With
    oSVG.appendChild oSVGElement

    oDoc.body.appendChild oSVG
    Debug.Print oDoc.all(0).outerHTML
End Sub

Sub NavigateBlank()
    ctlIE.Object.Navigate2 "about:blank"
    Do
        DoEvents
    Loop Until ctlIE.Object.ReadyState = 4
   Set oDoc = ctlIE.Object.Document
End Sub
```

Listing 1: Der wesentliche Code-Teil des Formulars **frmSVGTest** erzeugt im **Click**-Ereignis die Grafik

Die Zeichenfläche ist damit zwar erstellt, hat bislang jedoch noch keine definierten Abmessungen. Dies holt der folgende **With**-Block auf die Objektvariable nach. Um die Breite des **svg**-Elements einzustellen, können Sie zwei Methoden einsetzen. Die eine ist das Setzen des **width**-Attributs über die Methode **setAttribute**, die andere die Nutzung der eingebauten Eigenschaft **Width** der Objektklasse.

Letztes geschieht hier. Allerdings handelt es sich bei **Width** nicht etwa um einen Integer-Wert, sondern um einen speziellen zusammengesetzten **SVG**-Datentyp, dessen Eigenschaft **baseVal.Value** verwendet werden muss. Sie bekommt den Wert **300** Pixel zugewiesen. Ähnliches erfolgt für die Höhe der Fläche und **Height**. Und dann erhält die Zeichenfläche noch eine Hintergrundfarbe, die über das **Style**-Attribut **backgroundColor** mit einem hexadezimalen Grau-Wert versehen wird. Das **SVG**-Objekt wird erst in das Dokument geschrieben und damit sichtbar, wenn es am Ende der Routine über **appendChild** an den **Body oDoc.body** angehängt wird. Zuvor muss jedoch noch der **SVG**-Kreis erzeugt und an das **SVG**-Objekt selbst angehängt sein.

Deshalb generiert **createElementNS** nun ein Element **circle** und weist es als Objekt der Variablen **oSVGElement** vom **MSHTML**-Typ **SVGCircleElement** zu. Sie verweist nun auf ein abstraktes Kreis-Objekt, dessen Eigenschaften erst noch eingestellt werden müssen. Der folgende **With**-Block mit den **setAttribute**-Anweisungen übernimmt dies. Die Mittelpunktkoordinate wird mit **cx** und **cy** festgelegt, der Radius in **r** – ohne dezidierte Angabe von Einheiten geht der Browser dabei von Pixeln aus. Dem Element wird ein eindeutiger Bezeichner **Kreis** über die **id** spendiert, damit es später identifizierbar und damit sogar auch dynamisch bearbeitbar ist. Die Farbe des Kreises steht in **stroke**, die Linienbreite in **stroke-width** und die Füllfarbe im Attribut **fill**. Das Kreiselement wird schließlich mit **appendChild** an das **SVG**-Flächenelement **oSVG** angehängt, welches selbst dann in den **Body** geschrieben wird. Ab nun ist die Grafik tatsächlich sichtbar.

Obwohl wir nur mit **MSHTML**-Objekten hantieren und keinerlei String-Bearbeitung einsetzen, übersetzt sich das Ganze automatisch in gültigen **HTML**-Text. Die **Debug.Print**-Anweisung verdeutlicht, was dabei herauskommt und gibt im VBA-Direktfenster diesen String aus, der hier der Übersicht halber mit Einrückungen formatiert wurde:

```
<html>
  <head><meta http-equiv="X-UA-Compatible""/></head>
    <body>
      <svg xmlns="http://www.w3.org/2000/svg"
        style="background-color: rgb(224, 224, 224);"
        width="300" height="250">
          <circle id="Kreis"
            fill="red" stroke="darkblue" stroke-width="4"
            cx="50" cy="50" r="40" />
      </svg>
    </body>
</html>
```

SVG Knowledgebase

Es liegt weit außerhalb des Radius dieses Beitrags, alle **SVG**-Elemente mit ihren Parametern darzustellen. Empfehlung: Schauen Sie sich etwa auf der deutschen Seite **SELFHTML** um, die unter der Rubrik **SVG** mit vielen Beispielen erschöpfend Auskunft gibt. Dort sind die möglichen Elemente und ihre Attribute gut erläutert. Das Vorgehen zur Übersetzung der **HTML-Tags** nach VBA ist dann immer dasselbe. Sie erzeugen ein Element über die Funktion **createElementNS** unter Angabe des Element-Tags, weisen es einer Objektvariablen zu, die den generellen Typ **SVGElement** haben kann – oder auch schlicht **Object** –, und setzen dessen Eigenschaften am besten über **setAttribute**-Anweisungen innerhalb eines **With**-Blocks.

SVG-Filter

Das Formular **frmSVGTest** enthält noch eine weitere Schaltfläche **Komplexes SVG**, deren Betätigung zur Grafik in Bild 2 führt. Hier sind genau zwei SVG-Elemente angelegt, nämlich ein gelber Kreis mit grüner Umrandung und das Bild einer Tulpe aus einer JPG-Datei. Auf das Bild wird aber ein Weichzeichnenfilter angewandt, auf den Kreis ein Turbulenzfilter. Hier kommen neue Techniken ins Spiel.

Das Prinzip ist allerdings das gleiche, wie im vorigen Beispiel. Ein **SVG**-Element wird angelegt und an den

Bild 2: Einsatz von Filtern für die komplexere **SVG**-Grafik per VBA

Body des über **about:blank** erzeugten Dokuments gehängt. Das Vorgehen, um die Bilddatei in die Grafik zu bekommen, ist hier nicht trivial. Zwar könnte man das Attribut **href** des **image**-Elements mit dem absoluten Pfad zur Datei belegen, doch beim Verschieben der Datenbank und der Bilddatei **tulpe.jpg** in ein anderes Verzeichnis wäre der Bezug verloren. Deshalb wird aus der Bilddatei ein sogenanntes binäres **data-URI** erzeugt und dessen Inhalt dem Bildelement unmittelbar übergeben. Hintergrund dafür ist zusätzlich, dass sich so auch Dateien, die etwa in **OLE-Feldern** von in Tabellen abgespeichert sind, ebenfalls zur Ansicht bringen lassen. Das **data-URI** kommt dabei als **base64**-kodierter String in das Dokument.

Diese Funktionalität kam bereits im früheren Beitrag zum **Zeichnen und Malen mit HTML5** zum Einsatz.

Einen Teil der Ereignisprozedur **cmdCreate-Complex_Click** zeigt Listing 2. Die Bilddatei wird zuerst mit der **Open**-Anweisung in das Byte-Array **bin** eingelesen. Dann wird das leere **SVG**-Bildelement über das Tag **image** erzeugt.

Es bekommt die **id dataimage**, die Breite und Höhe **200px** (**width**, **height**), und die Bilddaten über **href** aus einem Hexadezimal-String, der mittels der Hilfsfunktion **EncodeBase64** im Modul **mdlHelper** aus dem Array erzeugt wird. Der **Internet Explorer** zeigt das Bild dann allerdings aus Sicherheitsgründen erst an, wenn zusätzlich die Eigenschaft **crossOrigin** auf **anonymous** gesetzt wird, weil sonst die Herkunft der Binärdaten für ihn ungeklärt ist.

Das Weichzeichnen kommt schließlich zustande, indem Sie der **filter**-Eigenschaft den Ausdruck **url(#blur)**

```
...
    Open CurrentProject.Path & "\tulpe.jpg" For Binary Access Read As #1
    ReDim bin(LOF(1) - 1)
    Get #1, , bin
    Close #1

    Set oSVGElement = oDoc.createElementNS("http://www.w3.org/2000/svg", "image")
    With oSVGElement
        .setAttribute "id", "dataimage"
        .setAttribute "width", "200"
        .setAttribute "height", "200"
        .setAttribute "href", "" & EncodeBase64(bin)
        .setAttribute "crossOrigin", "anonymous"
        .setAttribute "filter", "url(#blur)"
    End With
    oSVG.appendChild oSVGElement
...
```

Listing 2: Einlesen der Bilddatei in ein **Byte-Array** und Übergeben desselben an das **SVG-image**-Element

Objekt- und Feldnamen refaktorieren

Es kommt vor, dass man als Access-Entwickler mit der Weiterbearbeitung von Datenbanken betraut wird. Oft geschieht es dann, dass Objekt- und Feldnamen nicht den gängigen Konventionen entsprechen. Tabellennamen kommen ohne Präfix, Objekt- und Feldnamen enthalten Umlaute, Leerzeichen et cetera. Um dadurch entstehende Probleme zu vermeiden, können wir die Benennungen anpassen. Wie dies gelingt, zeigt der vorliegende Beitrag.

Die Tabelle und die Feldnamen des Beispiels aus Bild 1 hat jeder schon einmal gesehen. Die Tabelle wurde ohne Präfix benannt, hier würden wir uns ein führendes **tbl** wünschen. Das Primärschlüsselfeld hätten wir lieber ohne Unterstrich (wobei das Geschmackssache ist). Definitiv sollten wir aber Straße in Strasse ändern und im Feld **Telefon geschäftlich** wollen wir nicht nur den Umlaut durch die entsprechenden Vokale ersetzen und das Leerzeichen weglassen. Außerdem verwenden wir in der Regel Camel Case und schreiben das zweite und die folgenden Worte groß, hier also **TelefonGeschaeftlich**.

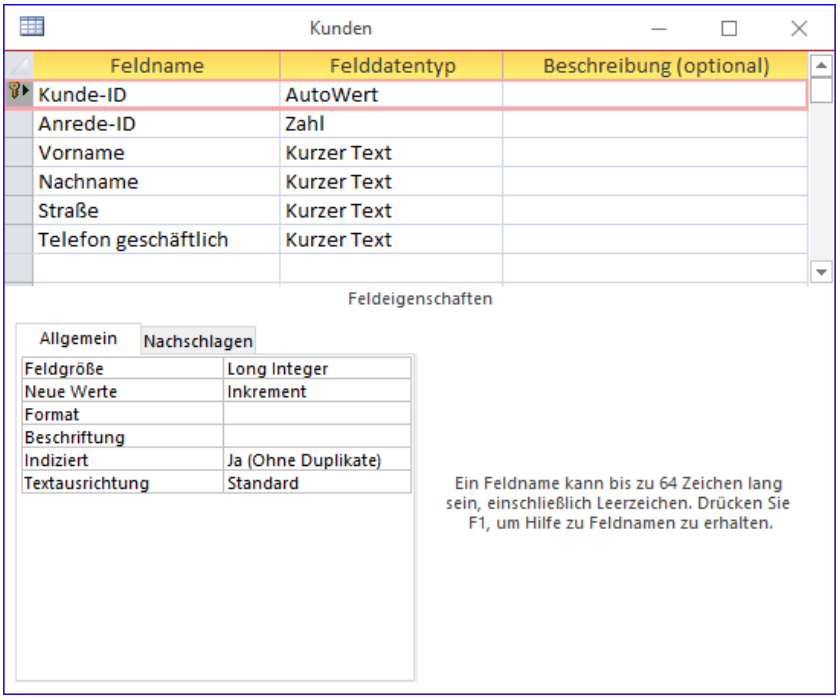


Bild 1: Tabellenname und Feldnamen sollten refaktoriert werden.

Das Ergebnis soll in unserem Fall also wie in Bild 2 aussehen.

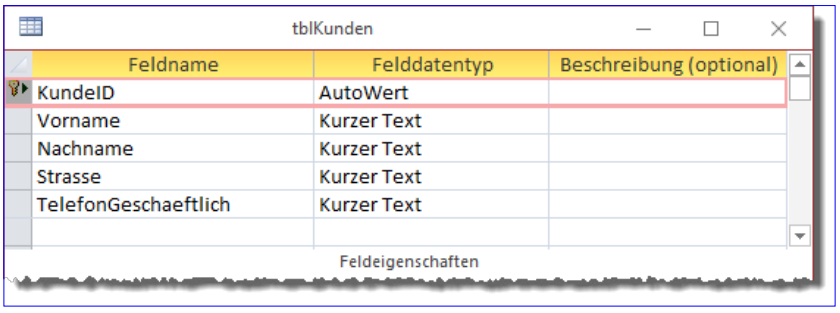


Bild 2: Tabelle mit Präfix im Namen und mit angepassten Feldnamen

Notwendige Änderungen finden

In unserem Beispiel ist es leicht, die zu ändernden Elemente zu finden. Bei einer Tabelle mit wenigen Feldern erledigt man den Job auch am einfachsten von Hand. Das bezieht sich allerdings nur auf die reine Änderung der Objekt- und Feldbezeichnungen – es gibt ja auch Referenzen auf diese Bezeichnungen in anderen Tabellen, Abfragen, Formularen, Berichten, Makros und auch im VBA-Code. Und wenn Sie aber eine große Datenbankanwendung mit vielen Tabellen anpassen wollen, haben Sie noch mehr zu tun. Es ist also eine Frage der Datenbankgröße, wann Sie es aufgeben, die Änderungen manuell zu erledigen und der Wunsch auftaucht, die Änderungen automatisch auszuführen. Wie das gelingt, soll

der vorliegende Beitrag zeigen. Vorher schauen wir uns aber noch an, welche Änderungen im Detail anfallen und wo diese sich überall auswirken.

Referenzen auf Objektnamen und Feldnamen

Die Namen von Tabellen und Feldnamen können bereits in anderen Tabellen auftauchen – nämlich dort, wo Sie Nachschlagefelder eingerichtet haben. Wenn wir unserer Anwendung also eine Tabelle namens **Anreden m/w** mit den Feldern **Anrede-ID** und **Anrede** finden, kann es sein, dass ein Fremdschlüsselfeld namens **Anrede-ID** in der Kunden-Tabelle vorliegt, welches die Anreden-Tabelle referenziert – das sieht dann im Entwurf wie in Bild 3 aus. Wenn wir nun auch die Bezeichnungen der verknüpften Tabelle ändern, dann müssen wir darauf achten, dass auch die Verweise angepasst werden. Zum Glück unterstützt uns Access hier – wie, schauen wir uns weiter unten an.

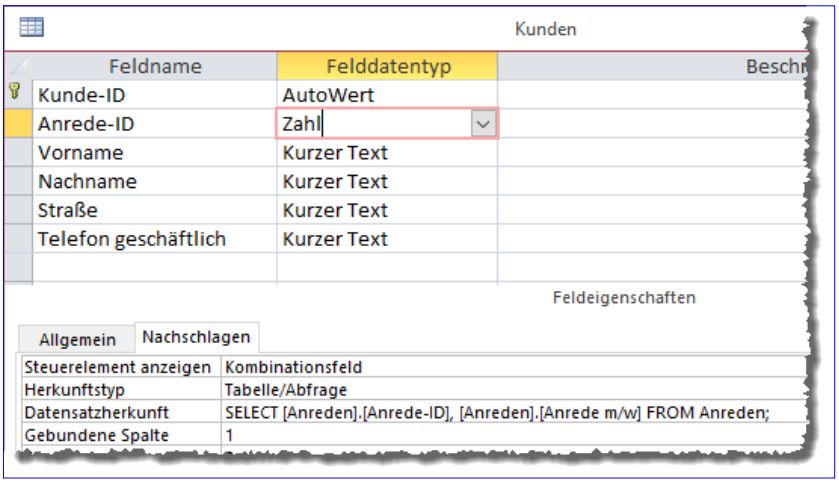


Bild 3: Verweis auf andere Tabelle im Tabellenentwurf

Aber auch in Abfragen, Formularen und Berichten sowie in Makros finden wir unter Umständen Verweise auf die Namen von Objekten und Feldern. Ob diese automatisch angepasst werden, schauen wir uns ebenfalls an.

Auf keinen Fall erfolgt eine automatische Anpassung im VBA-Code. Hier müssen wir selbst Hand anlegen. Damit wir hier keine Änderung auslassen, werden wir später ein wenig Code produzieren, der diese Aufgabe für uns übernimmt.

Die Objektnamen-Autokorrektur

Access hat schon viele Versionen eine Funktion namens Objektnamen-Autokorrektur. Diese sorgt dafür, dass Änderungen von Objekt- und Feldnamen direkt an Stellen übernommen werden, welche die betroffenen Objekte und Felder referenzieren.

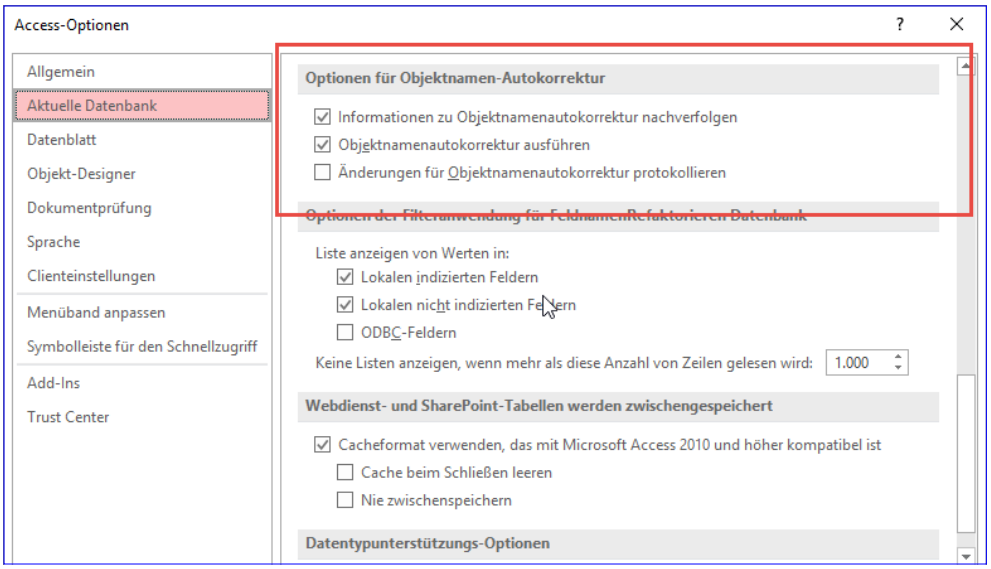


Bild 4: Einstellungen für die Objektnamen-Autokorrektur

Die Einstellungen für die Objektnamen-Autokorrektur nehmen Sie in den Access-Optionen vor. Dort finden Sie diese unter Aktuelle Datenbank im Bereich **Optionen für die Objektnamen-Autokorrektur** (siehe Bild 4). Wir aktivieren hier zusätzlich noch die

Option **Änderungen für Objektnamenautokorrektur protokollieren**.

Zu Testzwecken enthält die Beispieldatenbank einige Objekte, die auf die Tabelle und die enthaltenen Felder verweisen. Auf diese Weise können wir nach der Änderung prüfen, ob die Änderungen auch auf die referenzierenden Objekte übertragen wurden.

Wir verwenden dazu ein Formular namens **frmKunden**, das nur Daten der Tabelle **Kunden** anzeigt, ein Formular namens **frmKundenMitAnreden**, welches eine Abfrage mit den Daten aus den beiden Tabellen als Datenherkunft verwendet und ein Formular mit Unterformular, welches die Anreden im Hauptformular und die zugeordneten Kunden im Unterformular anzeigt.

Eine Abfrage namens **qryKundenNachAnrede** fasst die Daten der beiden Tabellen Kunden und Anreden zusammen.

Auf Makros wollen wir an dieser Stelle verzichten.

Außerdem schreiben wir ein paar VBA-Prozeduren, welche auf die Daten der Tabellen **Kunden** und **Anreden** zugreifen.

Bevor wir die Tabellen- und Feldnamen angepasst haben, haben wir die vorherige Version der Datenbank gesichert, und zwar unter dem Namen **FeldnamenRefaktorieren_Backup.accdb**.

Danach nehmen wir folgende Änderungen vor:

- Name der Tabelle **Anreden** auf **tblAnreden**
- Feld **Anrede-ID** wird **AnredeID**
- Feld **Anrede m/w** wird **Bezeichnung**
- Tabelle **Kunden** wird **tblKunden**
- Feld **Kunde-ID** wird **KundeID**
- Feld **Anrede-ID** wird **AnredeID**
- Feld **Straße** wird **Strasse**
- Feld **Telefon geschäftlich** wird **TelefonGeschaeftlich**

Nachdem die Änderungen an den Tabellen **Kunden** und **Anreden** durchgeführt wurden, sehen diese wie in Bild 5 aus. Durch die Änderungen an der Tabelle **Anreden**, die nun **tblAnreden** heißt, müssten auch Anpassungen am Fremdschlüsselfeld **AnredeID** der Tabelle **tblKunden** vorgenommen worden sein.

Nach der Änderung schauen wir uns den Entwurf der Tabelle **tblKunden** an und wechseln dort für das Feld **AnredeID** zur Registerseite **Nachschlagen** der Feldeigenschaften. Hier finden wir die korrekt geänderte Abfrage vor (siehe Bild 6).

Komplexere Ausdrücke

Wenn wir nun statt einer einfachen **SELECT**-Anweisung einen komplizierteren, von Hand geänderten Ausdruck für die Abfrage in der Eigenschaft **Datensatzherkunft** nutzen, funktioniert die Objektnamen-Autokorrektur in unseren Tests immer noch. Wenn Sie zum Beispiel einfach zwei Felder verknüpfen wie im folgenden Beispiel, wird die Abfrage ebenfalls ordnungsgemäß angepasst:

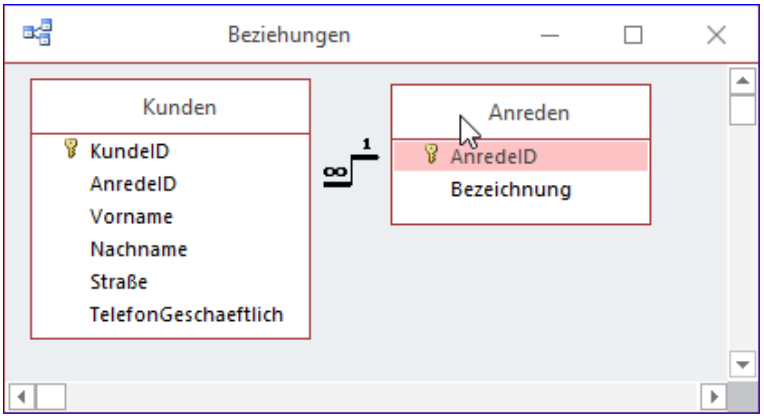


Bild 5: Die Tabellen nach der Änderung

```
SELECT [Anreden].[Anrede-ID], [Anreden].
[Anrede m/w] & Anreden.[Anrede-ID] FROM
Anreden;
```

Die Tabelle Objektnamen-Autokorrekturprotokoll

Da wir die Option Änderungen für Objektnamenautokorrektur protokollieren aktiviert haben, erstellt Access nun eine neue Tabelle namens Objektnamen-Autokorrekturprotokoll, in der es die einzelnen durch die Änderung verursachten Änderungen einträgt. Genaugenommen muss man es etwas anders formulieren: Die automatischen Änderungen werden nämlich nicht schon in die Tabelle eingetragen, sobald der Programmierer die auslösenden Änderungen vornimmt, sondern erst, wenn die von den automatischen Änderungen betroffenen Objekte erstmals nach der Änderung geöffnet werden.

Access prüft also wohl erst beim Öffnen der Objekte, ob es Änderungen in den zugrunde liegenden Objekten gibt und nimmt dann die Autokorrektur vor. Dies ist auch

Bild 6: Fremdschlüsselfeld nach der Änderung der verknüpften Tabelle

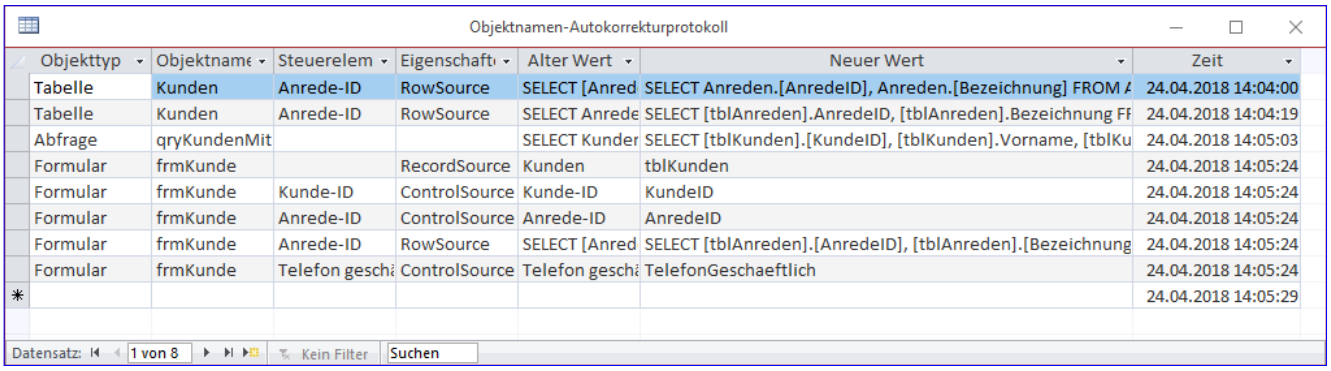


Bild 8: Änderungen durch die Objektnamen-Autokorrektur

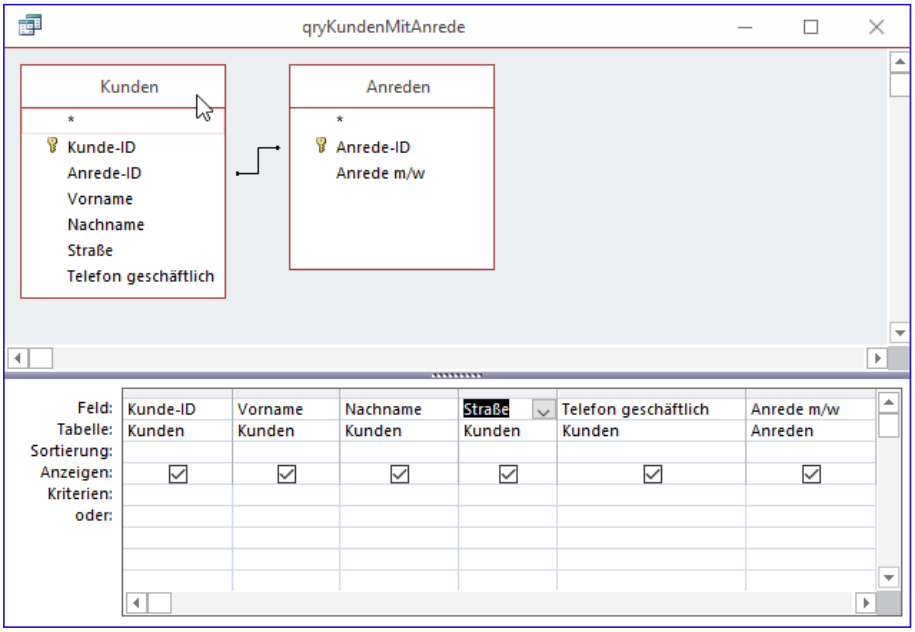


Bild 7: Die Abfrage **qryKundenMitAnrede** vor den Änderungen

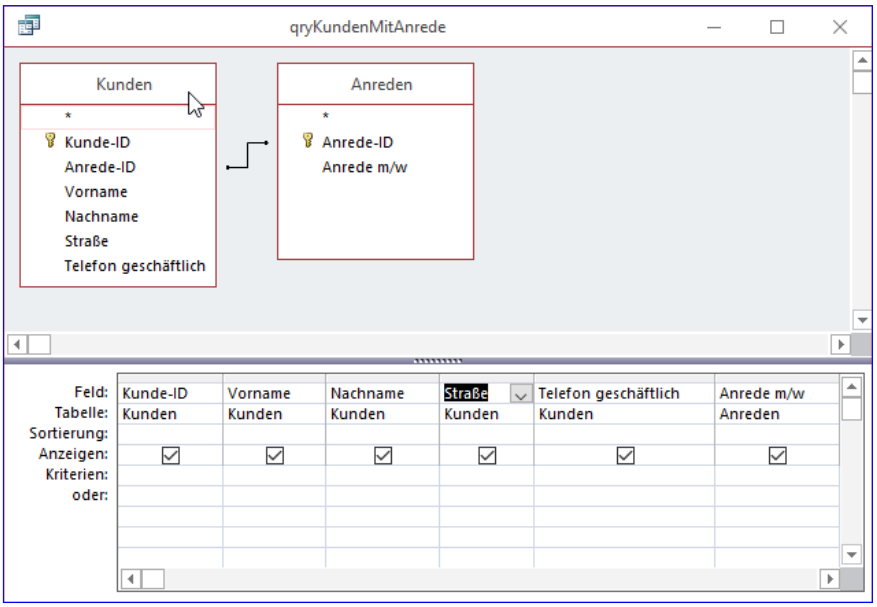


Bild 9: Die Abfrage **qryKundenMitAnrede** nach den Änderungen

der Grund, warum man bei Datenbanken, deren Performance nicht überzeugt, als eine der ersten Aktionen die Autokorrektur abschaltet.

Die Tabelle **Objektnamen-Autokorrekturprotokoll** sieht schließlich wie in Bild 8 aus.

Sie enthält bereits Einträge, die durch die nachfolgend beschriebenen Änderungen verursacht wurden.

Abfragen

Dann gehen wir weiter zur Abfrage **qryKundenMitAnrede**. Diese sah zuvor wie in Bild 7 aus.

Nachdem wir sie nach den Änderungen erneut geöffnet haben, finden wir dort ebenfalls die korrigierten Tabellen- und Feldnamen vor (siehe Bild 9).

Formulare

Auch die drei vorbereiten Formulare wollen wir uns anschauen. Beim ersten Formular namens **frmKunden** hat dies gut geklappt. Wie Bild 10 zeigt, wurde nicht nur die als Datenherkunft verwendet Tabelle von **Kunden** in **tblKunden** geändert, sondern auch die Namen der von den Steuerelementen referenzierten Felder.

Das zweite Formular namens **frmKundenMitAnrede**, das Sie in Bild 11

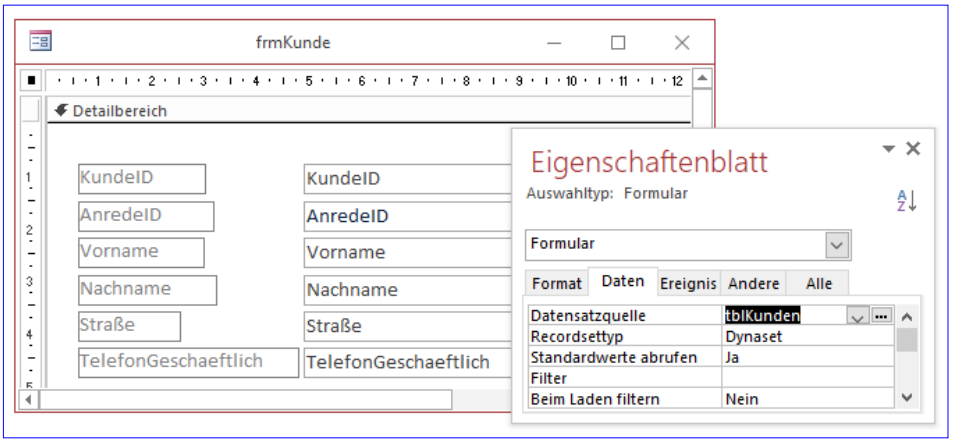


Bild 10: Formular mit geänderten Steuerelementinhalten und Quelltabelle

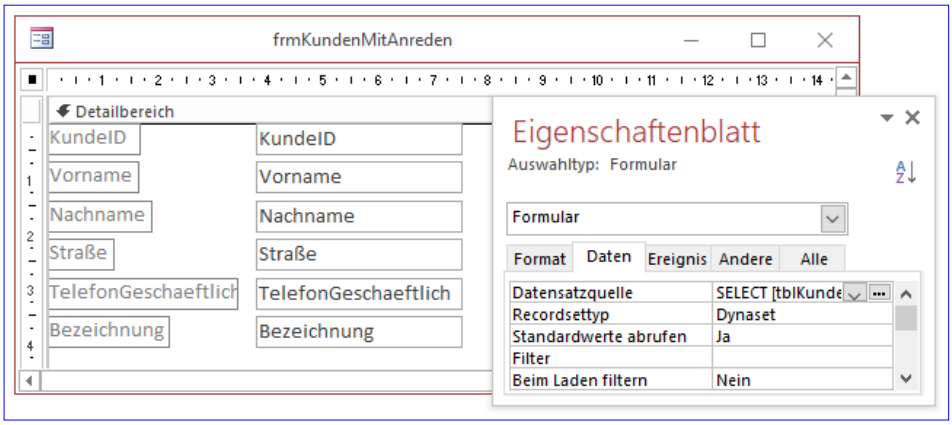


Bild 11: Auch mit einer Abfrage als Datenherkunft gelingt die Autokorrektur.

Vereinsverwaltung: Formulare, Teil 1

In den Beiträgen »Vereinsverwaltung: Von Excel zum Datenmodell« und »Vereinsverwaltung: Migration« haben wir uns um die Erstellung eines Datenmodells und die Migration bestehender Beispieldaten aus einer Excel-Tabelle gekümmert. Für die Bearbeitung der Daten aus den so erstellten und gefüllten Tabellen wollen wir nun eine Benutzeroberfläche programmieren. Der vorliegende Beitrag zeigt, wie wir die Formulare gestalten.

Ausgangsposition

In den beiden vorhergehenden Teilen dieser Beitragsreihe, nämlich **Vereinsverwaltung: Von Excel zum Datenmodell** (**www.access-im-unternehmen.de/1106**) und **Vereinsverwaltung: Migration** (**www.access-im-unternehmen.de/1107**) haben wir exemplarisch gezeigt, wie wir aus den vorliegenden Daten und Anforderungen ein Datenmodell für eine Vereinsverwaltung erstellen. Das Ergebnis finden Sie in Bild 1. Der Kern des Datenmodells ist die Tabelle **tblMitglieder**, um die herum wir eine Menge weiterer Tabellen angelegt haben. Diese dienen entweder als Lookup-Tabellen für die Fremdschlüsselfelder der Tabelle **tblMitglieder** oder nehmen andere Daten auf – zum Beispiel die Dauer der Vereinszugehörigkeit (**tblVereinszugehoerigkeiten**), die Altersklassen oder die Zusammenfassung von mehreren Mitgliedern zu einer Familie (**tblFamilienmitglieder**).

Vorbereiten der Feldbeschriftungen

Wir wollen ganz einfach mit einem Detailformular für die Tabelle **tblMitglieder** starten. Bevor wir damit beginnen, wollen wir noch die Feldbeschriftungen für die einzelnen Felder anpassen. Wenn wir nämlich die Felder einer Tabelle nach dem Zuweisen der Tabelle an die Eigenschaft Da-

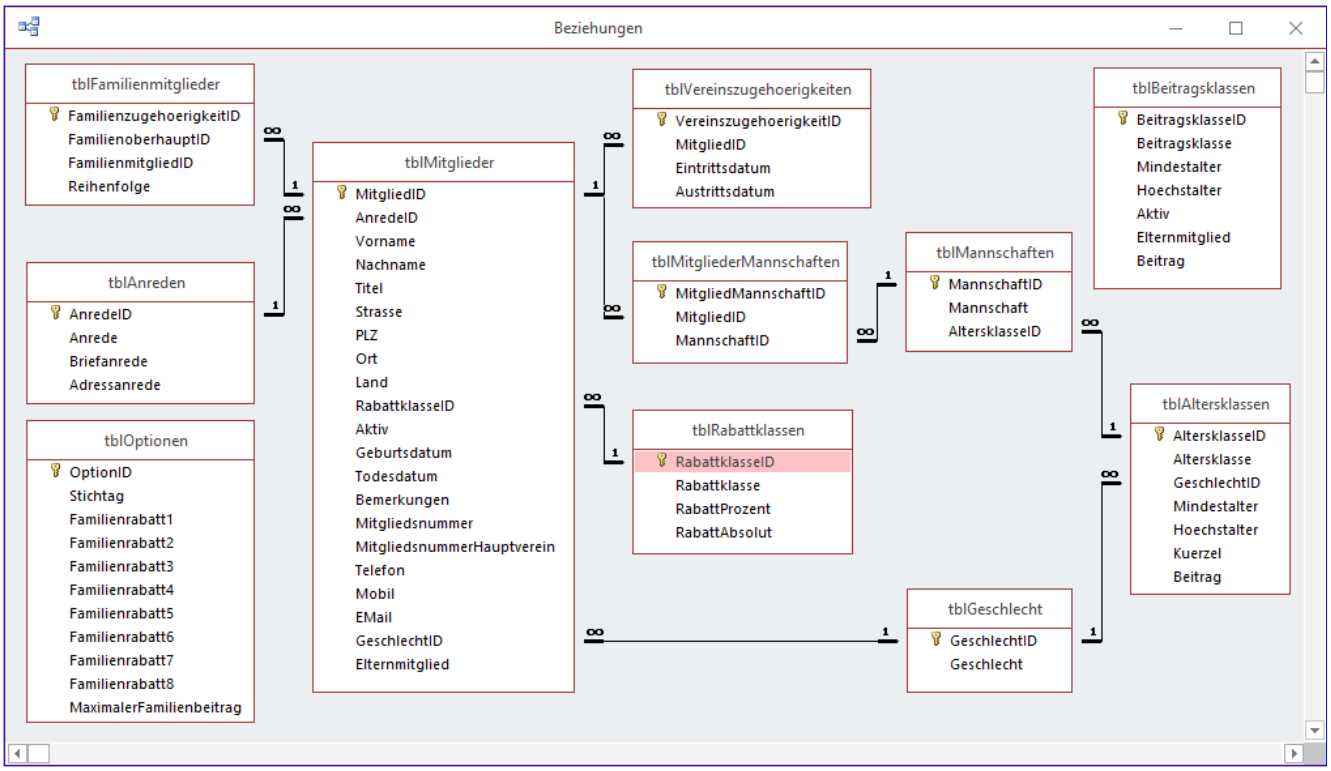


Bild 1: Datenmodell der Vereinsverwaltung

tenherkunft des Formulars aus der Feldliste in den Formularentwurf ziehen wollen, trägt Access immer den Feldnamen in das automatisch angelegte Bezeichnungsfeld ein. Das heißt, dass für die Anrede beispielsweise der Feldname **AnredeID** als Bezeichnungsfeld verwendet wird. Damit wir nicht immer die Bezeichnungsfelder anpassen müssen, stellen wir gleich den gewünschten Feldnamen in der Tabellendefinition ein, und zwar wie in Bild 2. Hier legen wir beispielsweise den Wert **Anrede** für die Eigenschaft **Beschriftung** fest. Auf die gleiche Weise gehen wir mit den folgenden Feldern vor:

- **Strasse** wird zu **Straße**
- **RabattklasseID** wird zu **Rabattklasse**
- **MitgliedsnummerHauptverein** wird zu **Mitgliedsnr. (Hauptverein)**
- **EMail** wird zu **E-Mail**
- **GeschlechtID** wird zu **Geschlecht**

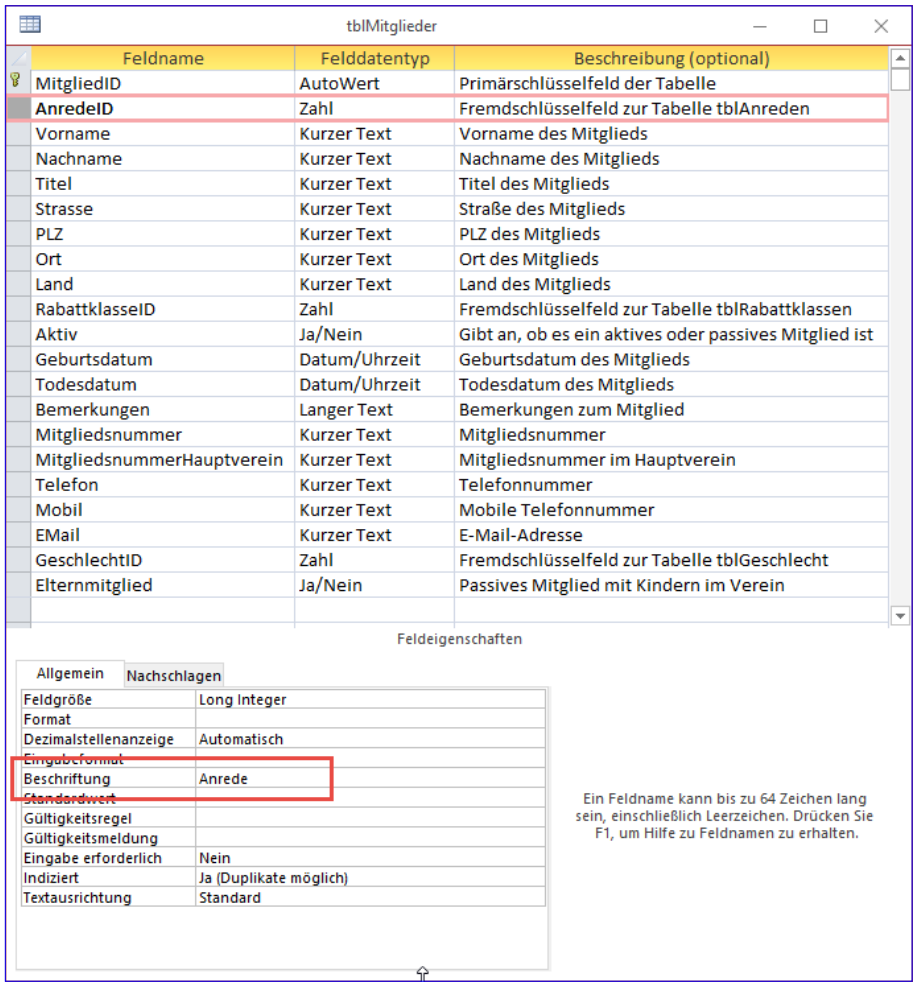


Bild 2: Anpassen der Feldbeschriftungen

Doppelpunkt aktivieren

Damit Access beim Hinzufügen der Felder aus der Feldliste in den Detailbereich des Formulars nicht nur die passenden Beschriftungen für die Bezeichnungsfelder einträgt, sondern diese auch noch um einen Doppelpunkt ergänzt, müssen Sie gegebenenfalls noch einmal Hand anlegen. Öffnen Sie dazu ein neues, leeres Formular und klicken Sie auf die Schaltfläche zum Hinzufügen eines neuen Textfeldes (nur anklicken, noch nicht einfügen!). Das Eigenschaftenblatt zeigt nun auf der Registerseite **Format** fast ganz unten die Eigenschaft **Mit Doppelpunkt** an. Diese stellen Sie auf den Wert **Ja** ein (siehe Bild 3).

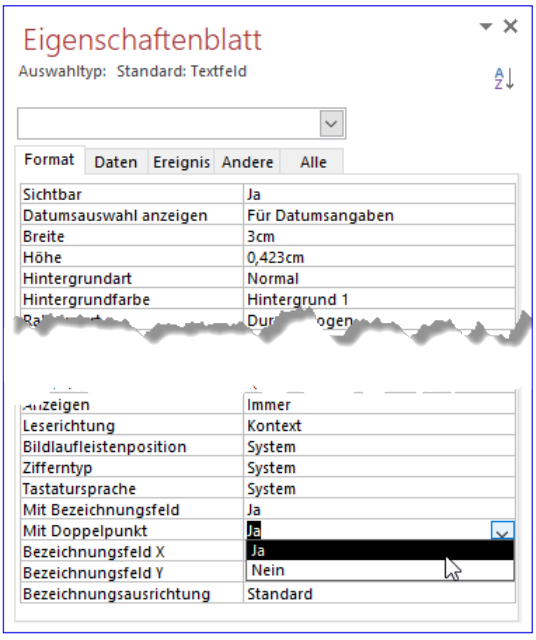


Bild 3: Aktivieren der Doppelpunkte für Beschriftungen

Speichern Sie das Formular nun unter dem Namen **Normal**. Prüfen Sie dann, ob in den Access-Optionen wie in Bild 4 diese Bezeichnung für die Eigenschaft **Formularvorlage** festgelegt ist. Wenn Sie nun neue Formulare anlegen, berücksichtigen diese die Einstellungen, die wir hier für den Steuerelementtyp **Textfeld** vorgenommen haben.

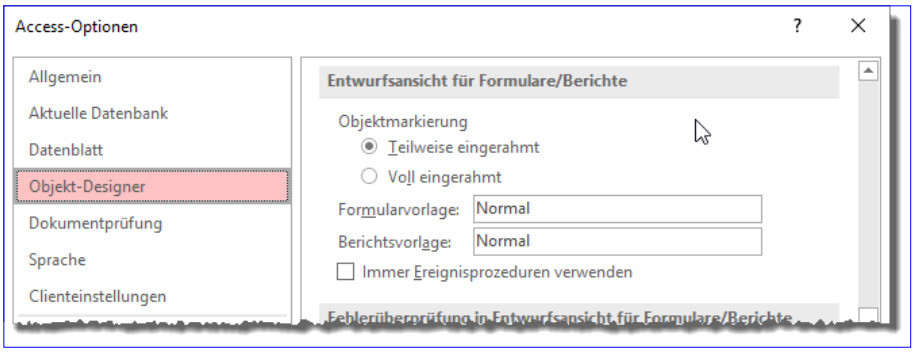


Bild 4: Bezeichnungen der Vorlagen für Formulare und Berichte

Formular für die Mitglieder anlegen

Nun legen wir ein neues, leeres Formular an und stellen seine Eigenschaft **Datenherkunft** auf die Tabelle **tblMitglieder** ein. Speichern Sie das Formular vorsichtshalber gleich unter dem Namen **frmMitglieddetails**. Nun aktivieren Sie die Feldliste mit dem Ribbon-Befehl **Entwurf|Tools|Vorhandene Felder hinzufügen**. Markieren Sie alle Einträge des Fensters **Feldliste** und ziehen Sie diese in den Detailbereich des Formularentwurfs (siehe Bild 5).

Hier tragen unsere Vorbereitungen nun Früchte: Sowohl die Beschriftungen der Bezeichnungsfelder entsprechen den von uns angegebenen Werten als auch die Doppelpunkte wurden wir gewünscht hinzugefügt.

Daten aus weiteren Tabellen hinzufügen

Bevor wir die bisher hinzugefügten Steuerelemente in sinnvollen Gruppen anordnen, schauen wir uns noch an, welche Daten aus den anderen Tabellen wir noch benötigen:

- **tblFamilienmitglieder**: Wir könnten in einem Unterformular die übrigen Familienmitglieder anzeigen, sofern diese vorhanden sind.
- **tblVereinszugehoerigkeiten**: Die Vereinszugehörigkeiten sollten auf jeden Fall in einem Unterformular angezeigt werden.
- **tblMitgliederMannschaften/tblMannschaften**: Auch die Zugehörigkeiten zu einer oder mehrerer Mannschaften könnte man in diesem Formular dokumentieren und auch bearbeiten.

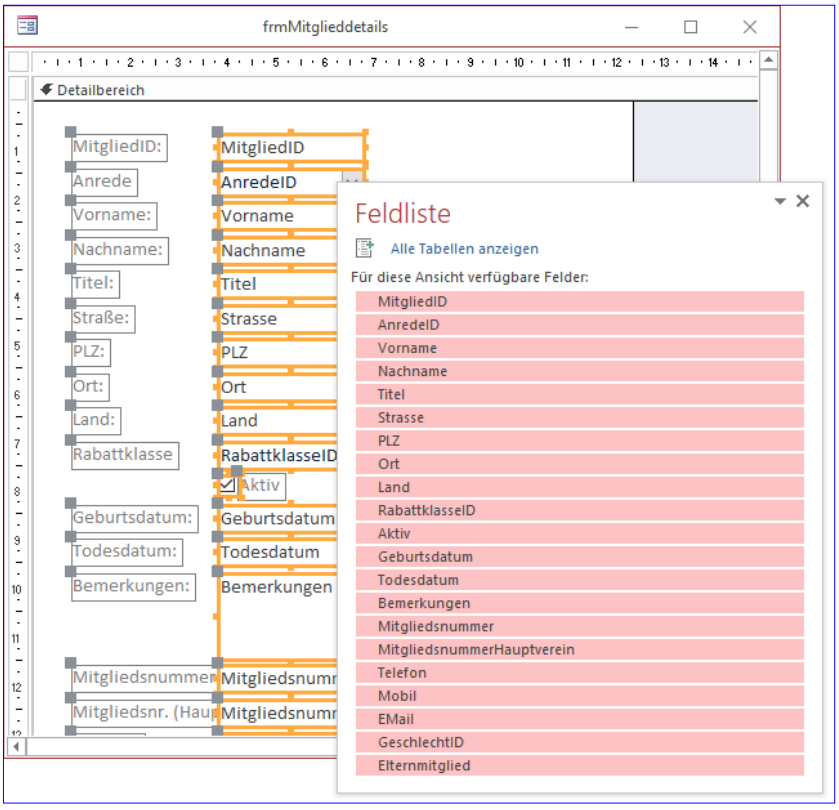


Bild 5: Hinzufügen der Felder zum Formular **frmMitglieddetails**

- **tblAnreden**, **tblRabattklassen**, **tblGeschlecht**: Die Werte dieser Felder werden bereits über Nachschlagefelder dargestellt.

Bei den Daten der Tabellen **tblFamilienmitglieder**, **tblVereinszugehoerigkeiten** und **tblMitgliederMannschaften** können wir festlegen, auf welche Art wir die Daten anzeigen wollen. Wir wollen in dieser Beispiellösung ein paar verschiedene Ansätze demonstrieren, aber die Ergonomie soll auch nicht zu kurz kommen. Schauen wir uns die Tabellen im Detail an.

Bearbeiten der Vereinszugehörigkeiten

Die Vereinszugehörigkeiten sind die einfachste Darstellung. Es handelt sich ja einfach nur um eine 1:n-Beziehung, wobei jedes Mitglied eine oder mehrere Vereinszugehörigkeiten aufweisen kann. Da es vermutlich nicht allzuoft geschieht, dass ein Mitglied den Verein verlässt und dann zurückkehrt, wollen wir diese Daten in einem Listenfeld anzeigen, dessen Einträge wir über einen Doppelklick bearbeiten können beziehungsweise durch entsprechende Schaltflächen neue Vereinszugehörigkeiten anlegen oder löschen können.

An dieser Stelle fällt auf, dass wir die Eintritts- und Austrittsdaten noch gar nicht aus der Excel-Tabelle in das neue Datenmodell übernommen haben. Das holen wir nun noch nach. Dabei unterstützt uns die Funktion **Eintrittsdatum** aus Listing 1. Diese öffnet zunächst ein Recordset auf Basis der Tabelle mit den aus der Excel-Datei importierten Mitgliedsdaten namens **Mitglieder**. In einer **Do While**-Schleife durchläuft sie alle Datensätze dieser Tabelle und ermittelt dabei zunächst den Wert des Primärschlüsselfeldes **MitgliedID** der Tabelle **tblMitglieder**, unter der das Mitglied importiert wurde. Um den entsprechenden Datensatz zu ermitteln, nutzen wir die **DLookup**-Funktion und suchen in der Tabelle **tblMitglieder** nach dem Datensatz, in dessen Feld **Mitgliedsnummer** der Wert gespeichert ist, den wir in der Excel-Datei in der Spalte **Mitgl-Nr** finden. Wenn wider Erwarten kein passender Datensatz gefunden wird, erhält **lngMitgliedID** den Wert **0** und die Anweisungen innerhalb der folgenden **If...Then**-Bedingung werden nicht ausgeführt. Stattdessen gibt die Routine eine entsprechende Meldung im Direktbereich des VBA-Editors aus.

Wurde das Mitglied jedoch gefunden, ermittelt die Prozedur den Wert der Spalte **Eintrittsdatum** und speichert diesen in der Variablen **datEintrittsdatum**. Während diese Spalte konsistent mit Datumswerten gefüllt war, müssen wir das eventuell vorhandene Austrittsdatum mühsam aus der Spalte **Bemerkungen** ermitteln. Dazu schreiben wir den Inhalt dieser Spalte in die Variable **strAustritt**. Es gibt in der Excel-Datei zwei Varianten von Austrittsdaten. Hat sich ein Mitglied abgemeldet, findet sich ein Eintrag wie »1.1.2018 abg.«, wenn ein Mitglied verstorben ist, ein Eintrag wie »am 1.1.2018 gest.«.

Wir bearbeiten beide Fälle in jeweils einer **If...Then**-Bedingung. Die erste prüft, ob **strAustritt** die Zeichenfolge **abg.** enthält. Falls ja, ersetzen wir mit der **Replace**-Methode die Zeichenfolge **abg.** durch eine leere Zeichenfolge und speichern das Ergebnis wieder in **strAustritt**. Anschließend entfernen wir alle führenden und folgenden Leerzeichen mit der **Trim**-Funktion. Die folgende **If...Then**-Bedingung prüft mit der **IsDate**-Funktion, ob **strAustritt** nun ein gültiges Datum enthält. Falls ja, landet dieser Wert nach Konvertierung mit **CDate** in der Variablen **datAustritt**, falls nicht, erscheint wieder eine entsprechende Meldung im VBA-Direktbereich.

Auf ähnliche Weise kümmern wir uns dann um verstorbene Mitglieder. Hier ersetzen wir allerdings die beiden Zeichenketten **gest.** und **am** durch leere Zeichenketten und verarbeiten die verbleibende Zeichenkette wie zuvor beschrieben.

Anschließend folgen je nach Inhalt von **datAustritt** zwei verschiedene SQL-Anfügeabfragen. Enthält die Variable **datAustritt** einen Wert größer **0**, was einem gültigen Datum entspricht, dann führen wir die erste **INSERT INTO**-Abfrage aus. Diese legt einen neuen Datensatz in der Tabelle **tblVer-**

einen neuen Datensatz in der Tabelle **tblMannschaften** mit der Bezeichnung aus **strAltersklasse** und dem Primärschlüsselwert dieser Altersklasse an und ermitteln danach den Wert des Primärschlüsselfeldes des neuen Datensatzes der Tabelle **tblMannschaften**. Dieser landet in der Variablen **lngMannschaftID**.

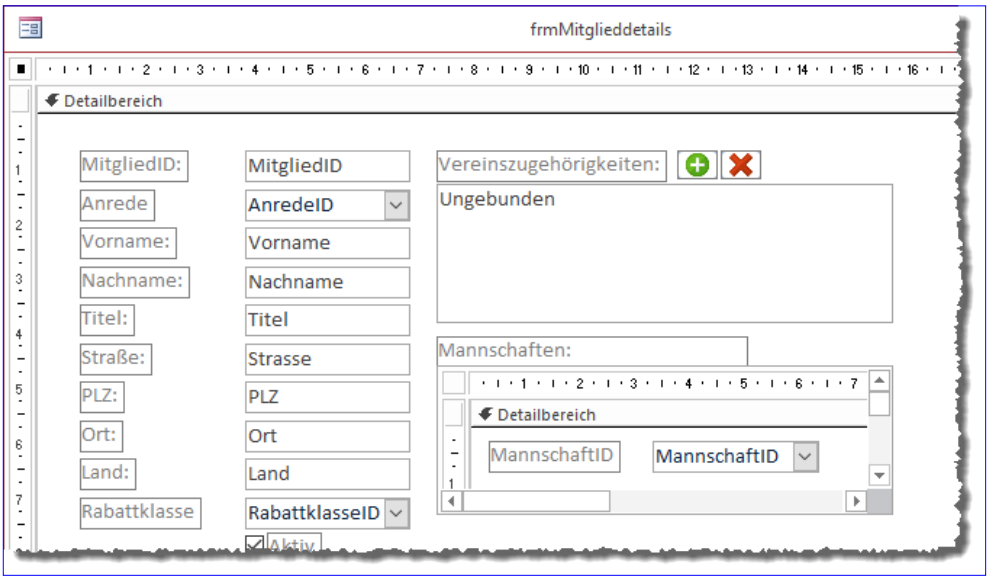


Bild 10: Mannschaften eines Mitglieds

Mit diesen Informationen können wir nun endlich den Eintrag in der Tabelle **tblMitgliederMannschaften** vornehmen, der die Zuordnung der Mitglieder zu den Mannschaften enthält. Die beiden Tabellen **tblMannschaften** und **tblMitgliederMannschaften** enthalten nun die Daten wie in Bild 9.

Unterformular für die Mannschaften eines Mitglieds

Um die Mannschaften eines Mitglieds im Formular **frmMitglieddetails** anzuzeigen, erstellen wir ein Unterformular namens **sfmMitglieddetailsMannschaften**, das die Tabelle **tblMitgliederMannschaften** als Datenherkunft verwendet und daraus das Feld **MannschaftID** im Detailbereich anzeigt.

Die Eigenschaft **Standardansicht** stellen wir auf **Datenblatt** ein. Wenn Sie dann das Unterformular schließen und es in den Detailbereich des Entwurfs des Formulars **frmMitglieddetails** ziehen, sieht das Ergebnis wie in Bild 10 aus.

In der Formularansicht können Sie dann die aktuell zugewiesene Mannschaft betrachten, ändern oder neue Mannschaften hinzufügen (siehe Bild 11).

Zusammenfassung und Ausblick

Das war der erste Teil der Beschreibung der Formulare der Vereinsverwaltung. Im folgenden Teil werden wir unter anderem das Formular **frmMitglieddetails** weiter ausbauen und ein Übersichtsformular zur Auswahl der Mitglieder liefern.

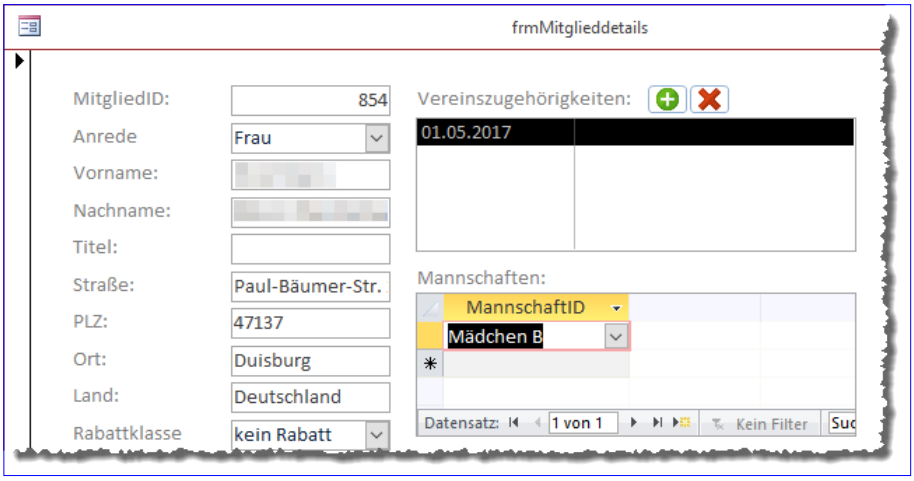


Bild 11: Mannschaften eines Mitglieds in der Formularansicht